SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Fabijan Josip Kraljić

Izrada baze podataka pomoću sustava SQLite

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Fabijan Josip Kraljić
Matični broj: 44013/15-R
Studij: Informacijski sustavi

Izrada baze podataka pomoću sustava SQLite

ZAVRŠNI RAD

Mentor:
Prof. dr. sc. Mirko Maleković

Varaždin, rujan 2018.

Fabijan Josip Kraljić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovim radom će biti prikazan SQLite SUBP, neke njegove specifičnosti, prednosti i nedostaci te koja bi bila njegova primjena. Osim toga, biti će izrađena baza podataka u istoimenom sustavu uz pomoć Navicat grafičkog alata za modeliranje baze podataka. Cijeli postupak izrade baze podataka u alatu Navicat biti će opisan i popraćen slikama, a kao primjer baze podataka uzeti će se osnovni oblik auto kuće.

Nakon što će se izraditi baza podataka, koristiti će se razvojno okruženje Android studio i programski jezik Java, za povezivanje i korištenje izrađene baze podataka, odnosno, izraditi će se mobilna aplikacija koja koristi novo napravljenu bazu podataka. Izrada aplikacije biti će popraćena isječcima koda, pomoću kojih će se opisati naredbe, koje su korištene prilikom izrade aplikacije.

Na samom kraju, ukratko će se prikazati primjena aplikacije te donijeti zaključak o samom sustavu SQLite, kao i o eventualnim, mogućim promjenama, a u cilju mogućeg poboljšanja aplikacije.

Ključne riječi: SQLite, Navicat for SQLite, DB Browser for SQLite, Android studio, Java, Mobilna aplikacija

Sadržaj

1. Uvod

Pohrana, zapisivanje, pamćenje podataka i informacija je uvijek bila bitna stvar, tako i danas u modernom i informatički okruženom svijetu. Još krajem 20. st. i početkom 21. st, tijekom tzv. doba Internet boom-a ili pojave tzv. Dot-com balona (engl. *bubble*) se pojavljivalo sve više i više Internet stranica koje su sadržavale velik broj informacija, a te informacije je bilo potrebno logično i strukturirano organizirati pa se i time pojavila velika potražnja za sustavima koji upravljaju bazom podataka, a jedan takav sustav je SQLite.

Sami cilj ovoga rada je izrada i prikaz baze podataka neke auto kuće ili auto salona, odnosno poslovanja iste. Baza podataka će biti izrađena u SQLite sustavu za upravljanje relacijskom bazom podataka uz pomoć alata Navicat. Razlog izrade ovakve baze podataka leži u vremenskoj prirodi, tj. zaposleniku i/ili korisniku kojem bi posao trebao biti olakšan, a time i ubrzan. Tako će biti moguće unositi i upravljati novim vozilima u voznom parku, upravljati njihovom lokacijom na imanju auto kuće, upravljati poslom poput servisa, pranja i slično, itd. Baza podataka će biti povezana s mobilnom aplikacijom, odnosno Android uređajem, koja će se realizirati uz pomoć razvojnog okruženja Android studio.

Kroz nekoliko poglavlja će biti prikazan sam sustav SQLite, njegove prednosti i nedostaci, te usporedba s drugim sustavima. Također, bit će prikazan i pojašnjen proces izrade baze podataka, tj. kako se uz pomoć grafičkog alata izrađuje baza podataka te koje naredbe stoje iza alata, prikazat će se kako se kreiraju okidači, a na kraju će biti prikazano povezivanje baze podataka s aplikacijom te njezino korištenje. Uz sav tekst i opise su priložene slike i isječci koda koji upotpunjuju napisane informacije.

2. SQLite

SQLite sustav za upravljanje bazom podataka (SUBP) je biblioteka koja je samostojeća, neovisna o drugim komponentama, biblioteka kojoj nije potreban server, niti nikakve dodatne konfiguracije, te biblioteka koja je transakcijskog tipa. Sami kod SQLite-a se nalazi na javno dostupnoj domeni, tj. izvorni kod je dostupan svima i to besplatno u bilo kakve svrhe, bilo to komercijalne ili privatne. SQLite sustav je jedan od najrasprostranjenijih sustava za upravljanje bazom podataka, koji se koristi sve od malih projekata do puno većih i značajnijih, a neke od tvrtki koje koriste SQLite su Adobe, Airbus, Apple, Facebook, Google, itd. (SQLite, 2018.).

2.1. Prednosti SQLite SUBP-a

Neke od prednosti SQLite sustava su navedeni u tekstu iznad. Prvo i ponajviše važno je to što je SQLite besplatan te je svima dostupan. Svatko tko radi na nekom projektu, velikom ili malom, a ponajviše nekom malom projektu će koristit SQLite kako bi izbjegao nepotrebne troškove. Iduća bitna prednost je ta da je SQLite biblioteka male veličine, tek 500KB ovisno o okruženju u kojem se koristi, a ona sama radi i upravlja svime te nisu potrebne nikakve druge biblioteke ili nadogradnje što znači da je pogodan za razne mobilne uređaje (SQLite, 2018.). SQLite biblioteka osim što je mala i jednostavna i dalje je vrlo sposobna, npr. može upravljati podacima sve do 2 terabajta (H3RALD, 2018.), što je i više nego dovoljno za bilo koju aplikaciju ili web stranicu te zbog toga većina web stranica koristi upravo SQLite. Osim navedenog, SQLite koristi većinu SQL 92 standarda, tako da je jednostavno izvršiti sve potrebne upite. Kao zadnje, SQLite nema odvojeni proces na nekom serveru, tj. podaci se čitaju i pišu direktno s diska iz jedne jedine datoteke u kojoj je zapisana cjelokupna struktura baze podataka te svi podaci koji su sadržani unutar nje. Uglavnom, SQLite je manji, jednostavniji, brži, prenosiv i prilagodljiv je.

2.2. Nedostaci SQLite SUBP-a

Na prvi pogled bi se reklo da je SQLite sustav idealan te da bi ga svi trebali koristiti, no to nije slučaj. Tako npr. kod serverskih aplikacija, odnosno kod slučaja gdje imamo veliki broj klijentskih programa koji šalju SQL upite istoj bazi podataka putem iste mreže tada može doći do značajnog kašnjenja zbog same prirode mreže (SQLite, 2018.) . U prijevodu SQLite se nebi trebao koristiti u slučajevima gdje će se istoj bazi podataka pristupati direktno i istovremeno s više računala putem neke mreže. Još jedan scenarij u kojem bi se SQLite trebao izbjegavati je kod web stranica s velikim prometom.

Za većinu web stranica to nije problem, ali ako se sa web stranice često zapisuju podaci u bazu podataka ili ako web stranica zahtjeva nekoliko servera, onda SQLite nije idealan te bi se tada trebali koristit profesionalni sustavi za upravljanje bazom podataka. Idući nedostatak je taj da su svi podaci zapisani u jednoj datoteci, a SQLite podržava baze do nekoliko desetaka terabajta (SQLite, 2018.), a u nekim slučajevima postoji potreba za još većim kapacitetom baze podataka. Osim toga, veliki je rizik od gubitaka podataka ili krađe budući da su svi podaci dostupni na jednom mjestu. Kada je riječ o velikoj količini podataka, onda je bolje te podatke raširiti na više manjih dijelova unutar servera radi redundantnosti, brzine i sigurnosti te je njima lakše upravljati putem klijentsko/serverskog sustava za upravljanje bazom podataka. Nadalje, SQLite dopušta samo jedno pisanje u određeno vrijeme, a u nekim slučajevima je potrebno vršiti više pisanja istovremeno. Zadnji nedostaci SQLite-a su ti da ne podržava *Right* i *Full Outer Join*, ne podržava potpune naredbe *Alter Table* i *Trigger*, pisanje u Poglede i ne podržava *Grant* i *Revoke*. (SQLite, 2018.)

2.3. Usporedba s drugim SUBP-ovima

Za sljedeću usporedbu će se uzeti dva sustava, MySQL i PostgreSQL, koje će se suprotstaviti SQLite sustavu kako bi se još bolje primijetile prednosti i nedostaci promatranog sustava. Osim što će za svaki sustav biti prikazani njegovi prednosti i nedostaci, također će biti navedeni stvarni slučajevi kada se koji sustav koristi.

2.3.1. Tipovi podataka

SQLite u odnosu na MySQL i PostgreSQL ima znatno manji broj podržanih tipova podataka, tako npr. za rad s brojevima kod SQLite raspolažemo tek s *Integer* i *Real* tipom podataka, dok kod druga dva sustava ima nekoliko tipova podataka pomoću kojih se može upravljati nad brojevima (npr. *Smallint*, *Bigint*, *Decimal*, *Bigserial* itd.). Za rad sa znakom ili znakovnim nizovima u SQLite-u raspolažemo tek s *Text* i *Blob* tipom podataka, a u MySQL-u raspolažemo s *Char*, *Varchar*, *Text*, *Blob* i različitim varijacijama istih, dok kod PostgreSQL raspolažemo s *Character*, *Character Varying*, *Text*, itd. Za upravljanje vremenskim tipom podataka kod SQLite-a ne postoji određeni tip podataka namijenjen u te svrhe te je potrebno programski urediti podatak u obliku teksta i u tom obliku ga pohraniti, dok MySQL i PostgreSQL raspolažu s *DateTime*, *Timestamp* i *Time*. Zadnji osnovni tip podataka je *Bool*, također SQLite ne raspolaže s takvim tipom podatka, već koristimo *Integer* pomoću kojeg onda programski riješimo potrebni problem. MySQL i PostgreSQL raspolažu s *TinyInt* i *Boolean*. Osim gore navedenih tipova podataka postoje i drugi specijalizirani tipovi podataka kojima raspolaže samo PostgreSQL, te neki drugi SUBP-ovi. To su *Macaddr*, *Money*, *Polygon*, *XML*, itd. koji mogu u nekim slučajevima biti vrlo korisni (DigitalOcean, 2018.).

2.3.2. Brzina rada

Kao što je rečeno u prethodnim odlomcima, SQLite pohranjuje cjelokupnu bazu podataka u jednu datoteku, što znači da je za većinu scenarija, malih i srednje velikih baza podataka pogodan za pretraživanje i čitanje.

Isto tako brz je i MySQL budući da se on odriče nekih standarda kako bi se postigle željene performanse, također je skalabilan na veće sustave gdje će i dalje brzo raditi. Na drugoj strani, PostgreSQL je previše napredan i složen, te za jednostavno čitanje podataka nije previše praktičan, pa je za takve zadatke bolje koristiti ili SQLite ili MySQL. Što se pisanja tiče, vrlo je jednostavno, SQLite podržava samo jedno pisanje u bazu u nekom trenutku, što uvelike utječe na same performanse, dok MySQL i PostgreSQL podržavaju više pisanja u nekome trenutku (DigitalOcean, 2018.).

2.4. Slučajevi korištenja

Iz viđenog je lako zaključiti da SQLite nije tu kako bi postao glavni SUBP već je tu da nadomjesti druge sustave koji u nekim slučajevima nebi bili praktični. Primjerice, kod mobilnih uređaja gdje je dostupno manje resursa i gdje je jednostavnost i praktičnost na prvom mjestu, ćemo izbjegavati znatno kompliciranije sustave poput MySQL-a ili PostgreSQL-a (Kreibich, 2010, str. 9).

SQLite se također može koristiti kao neka vrsta formata za dokument. Tada nije potrebno kreirati novi format, već se razne instance iz baze podataka mogu koristiti za prikaz dokumenta. U nekim slučajevima se ne treba koristiti aspekt relacijske baze podataka već samo osnovna pohrana podataka, budući da je SQLite transakcijskog tipa, svi ulazno/izlazni podaci su zaštićeni, odnosno nema gubitka podataka ili korupcije (Kreibich, 2010, str. 10).

Podaci SQLite baze podataka su dostupni na više platforma, što omogućuje laku migraciju podataka. Bazu podataka je lako premještati s jedne na drugu aplikaciju ili ispitati za pogreške, budući da to omogućuje sam format baze podataka. Format također omogućuje korištenje jednostavnih i javno dostupnih alata za upravljanje bazom podataka, što znači da je održavanje baze podataka vrlo lako (Kreibich, 2010, str. 11).

Vrlo bitna značajka SQLite-a je što omogućuje kreiranje baza podataka u radnoj memoriji, što je vrlo korisno kod temporalnih baza podataka koje ne zahtijevaju trajnu pohranu podataka, već se stvaraju po potrebi. Tada se koristi neki konvencionalni SUBP za dohvat podataka koji se onda pohranjuju u memoriju u SQLite bazu podataka iz koje se krajnji vidljivi podaci čitaju (Kreibich, 2010, str. 11).

3. Izrada baze podataka

Za svrhe ovog rada, kao baza podataka će poslužiti osnovni primjer jedne auto kuće ili nekog autosalona. Tako će baza imati mogućnost stvaranja vozila, upravljanje lokacijom samog vozila, upravljanje radnim nalozima te zaposlenicima. Samu bazu je moguće dodatno proširiti i detaljizirati kako bi se u budućnosti dodale neke druge funkcionalnosti no za svrhe ovog rada i preglednosti biti će prikazan sustav s osnovnim funkcionalnostima.

Baza podataka autokuće će sadržavati 23 tablice i 4 okidača (engl. *trigger*). Tablice su „Marka“, „Model“, „Motorizacija“, „DostupnaMotorizacija“, „Gorivo“, „Oblik“, „Boja“, „Oprema“, „Vozilo“, „OpremaVozila“, „Premjestaj_vozila“, „Lokacija“, „Lokacija_vozila“, „Adrese_autokuce“, „Uloga“, „Osoba“, „Zaduzene_osobe“, „Radni_nalog“, „Usluga“, „Drzava“, „Grad“, „Adresa“. Što se okidača tiče, oni su sljedeći: „Brisanje dostupnih motorizacija po određenoj izvedbi“, „Izmjena broja slobodnih mjesta na lokaciji“, „Provjera slobodnih mjesta prilikom unosa premjestaja vozila“ i „Unos nove lokacije vozila nakon premjestaja“.

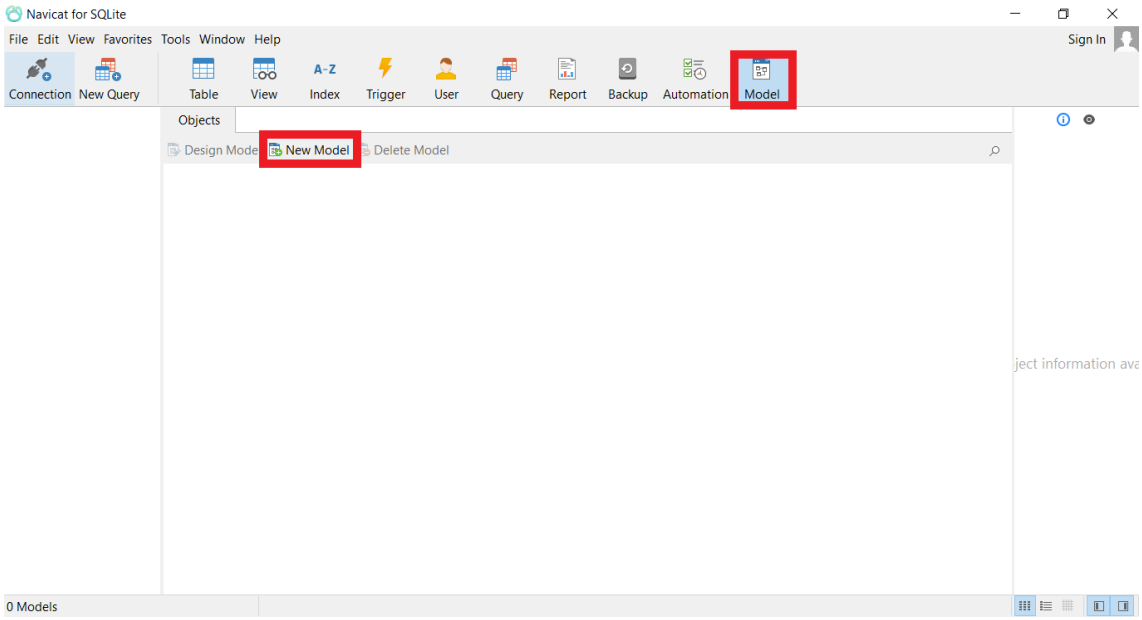
Prilikom kreiranja baze podataka će se koristiti dva alata, Navicat i DB Browser, no nije potrebno koristiti oba alata ili bilo koji od njih, postoje i razni drugi alati, tako npr. umjesto Navicat-a možemo koristiti DBeaver ili HeidiSQL i sl.

3.1. Kreiranje tablice baze podataka

Za kreiranje tablica, odnosno baze podataka koristit će se grafički alat Navicat. Postoje razne izvedbe kao što su Navicat za MySQL, za PostgreSQL, SQL Server i razne drugi SUBP, te postoji Premium verzija koja objedinjuje razne sustave unutar jedne aplikacije (Navicat, 2018.). Za svrhe ovog rada je korištena Trial (besplatna verzija) verzija alata namijenjena SQLite-u. SQLite bazu podataka je moguće kreirati putem komandne linije ili *Command Promt* sučelja u koju se upisuju SQL naredbe kako bi se kreirala baza podataka. Ovaj pristup će se izbjeći jer je dosta nepregledan i spor, što ne znači da je beskoristan, već ima svoje svrhe. Takav pristup nam daje potpunu kontrolu nad bazom podataka i ne ovisimo o nekoj trećoj strani pa u svakom trenutku znamo što se događa s našom bazom podataka ili koje se naredbe koriste. Kao što je rečeno, Navicat je grafički alat za modeliranje baze podataka, te radi na „*drag and drop*“ principu, no ima mogućnosti pisanja SQL naredbi kako bi korisnik i dalj imao mogućnost specificiranja točne naredbe. Prije nego što se baza podataka izradi, prvo je potrebno specificirati njezin model[1] na konceptualnoj razini.

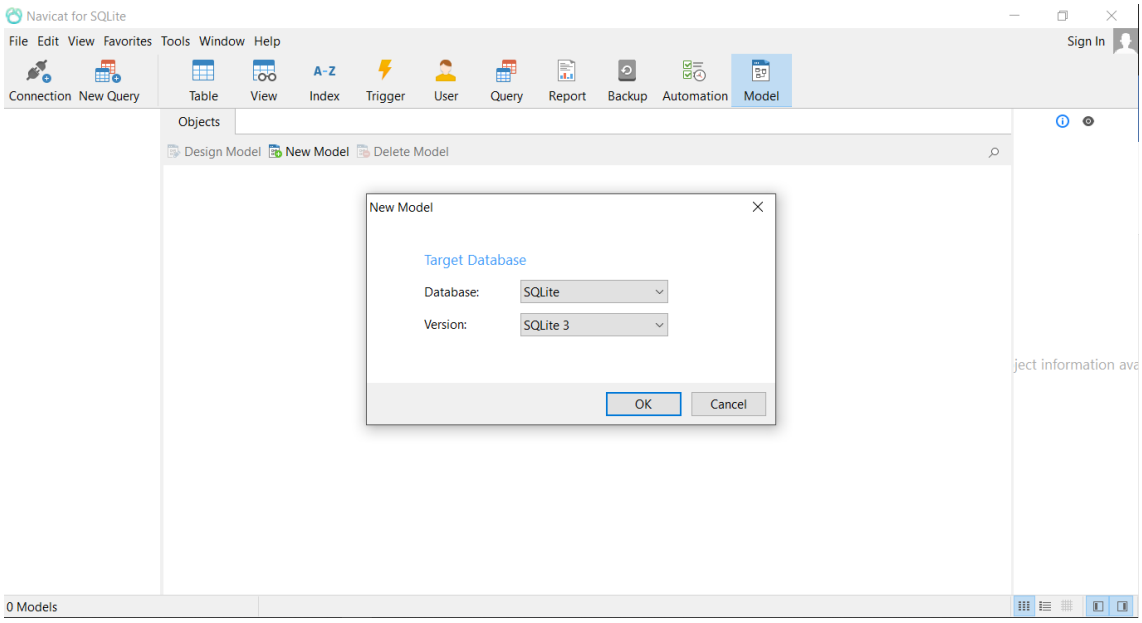
[1] Model – model podataka koji određuje logičku strukturu neke baze podataka te utvrđuje na koji se način podaci pohranjuju, kako su organizirani i kako se njima manipulira.

Pri vrhu prozora, na alatnoj traci se nalazi ikona ispod koje piše model te se odabere ta opcija. Lokacija ikone je prikazana na slici ispod (*Slika 1*) unutar crvenog pravokutnika. Nakon toga se odabere *New model,* koji je također označen crvenim pravokutnikom.



Slika 1. Stvaranje novog modela (vlastita izrada)

Pritiskom na *New model* otvara se novi prozor (*Slika 2*) u kojem se odabere vrsta baze podataka, odnosno verziju, tako bi npr. kod Premium verzije mogli birati između MySQL, PostgreSQL, Oracle, itd. budući da je ova verzija namijenjena samo za SQLite, onda se odabere jedna jedina opcija i verzija sustava, a to je uglavnom ona najnovija, SQLite 3 u ovome slučaju, a nakon toga se pritisne gumb OK.

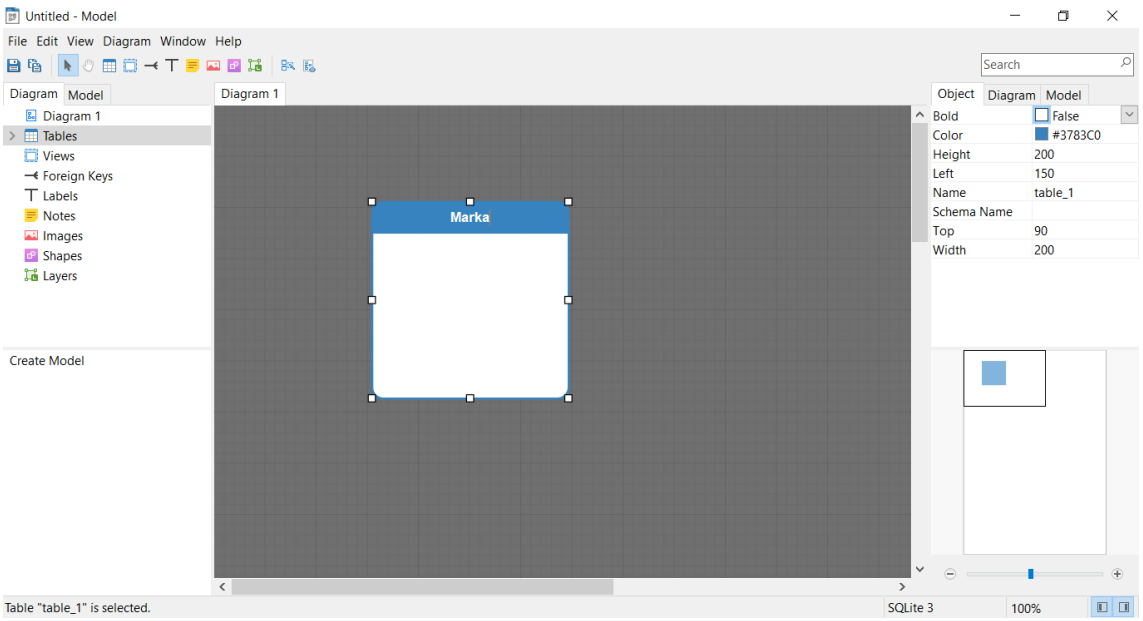


Slika 2. Odabir vrste i verzije modela (vlastita izrada)

Otvara se novi prozor (*Slika 3*) gdje se kreira model, a pri prvom pokretanju uvijek piše *Untitled – Model*, tek se kasnije pri prvom spremanju specificira naziv. Na alatnoj traci i kučici desno uz rub nalaze se razne ikone, kao što su ikone *tables, views, foreign keys, labels, notes* itd., pomoću kojih će se kreirati tablice, povezivati ih, prikazivati podatke i sl.

Slika 3. Prozor Modela (vlastita izrada)

Tek sad slijedi kreiranje samog modela, a to se radi tako da se iz alatne trake odabere ikona tablice (*Tables*), pritisne se lijevim klikom na površinu čime se kreira sama tablica. Tablica se zatim može imenovat te joj se mogu specificirati atributi.

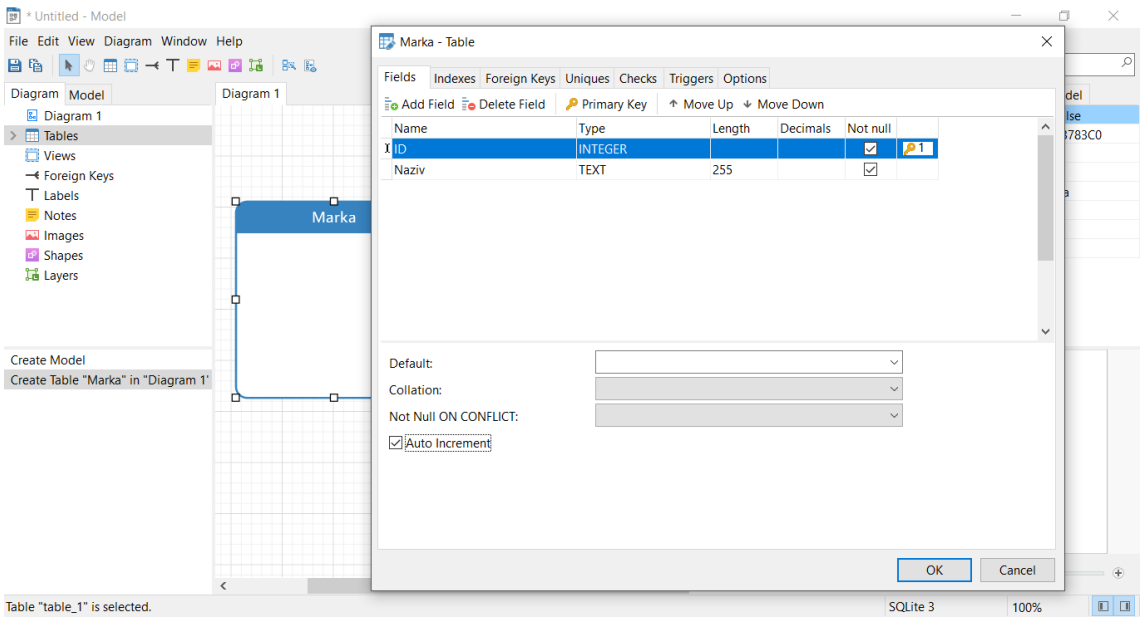


Slika 4. Stvaranje tablice (vlastita izrada)

Kako bi se upisali određeni atributi za odabranu tablicu, potrebnu je na istu tablicu dva puta kliknuti lijevim klikom miša ili pritisnuti desni klik miša na određenu tablicu kojim se otvora izbornik iz kojeg se odabere *Design table.* U oba se slučaja otvori novi prozor u kojem je moguće dodavati nove atribute tablici, specificirati tip podatka atributa, specificirati primarni/e ključeve, vanjske ključeve, okidače, indekse, razna ograničenja, itd.

Pod stupcem *Name* upiše se ime/na stupca odabrane tablice, pod stupcem *Type* se odabere jedan od tipova podataka, tako se za identifikacijski stupac (primarni ključ) često odabere *Integer,* iako može biti i *Text* tipa podatka, te se postavi da je primarni ključ. Primarni ključ[2] se odredi na način da se odabere novo upisani red i pritisne se ikona s nazivom *Primary key.* Ukoliko želimo da se primarni ključ (broj) povećava prilikom unosa novih podataka u tablicu, onda je potrebno odabrati *Auto Increment* na dnu prozora te se tada prilikom svakog novog unosa u tu tablicu primarni ključ poveća za jedan.

Svi ostali stupci, koji nisu identifikacijski, se kreiraju na jednaki način, odabere se tip podatka (*Text*, *Real*, *Integer*, *Bloob*), pod *Lenght* se postavi maksimalna dužina podatka koji se može upisati, npr. 20 znakova ili brojeva, a pod *Decimals* se postavi broj decimala. Kao zadnje, za pojedine stupce se može odrediti je li u njih potrebno unijeti neku vrijednost ili njihova vrijednost može ostati nepoznata. To se odredi označavanjem kućice *Not null*, a ako je ta kućica označena, onda prilikom unosa podataka u tu tablicu pod tim stupcem je potrebno unijeti podatak, inače se unos odbacuje. Sve iznad navedeno je prikazano na slici ispod (*Slika 5*).



Slika 5. Dodavanje i uređivanje atributa tablice (vlastita izrada)

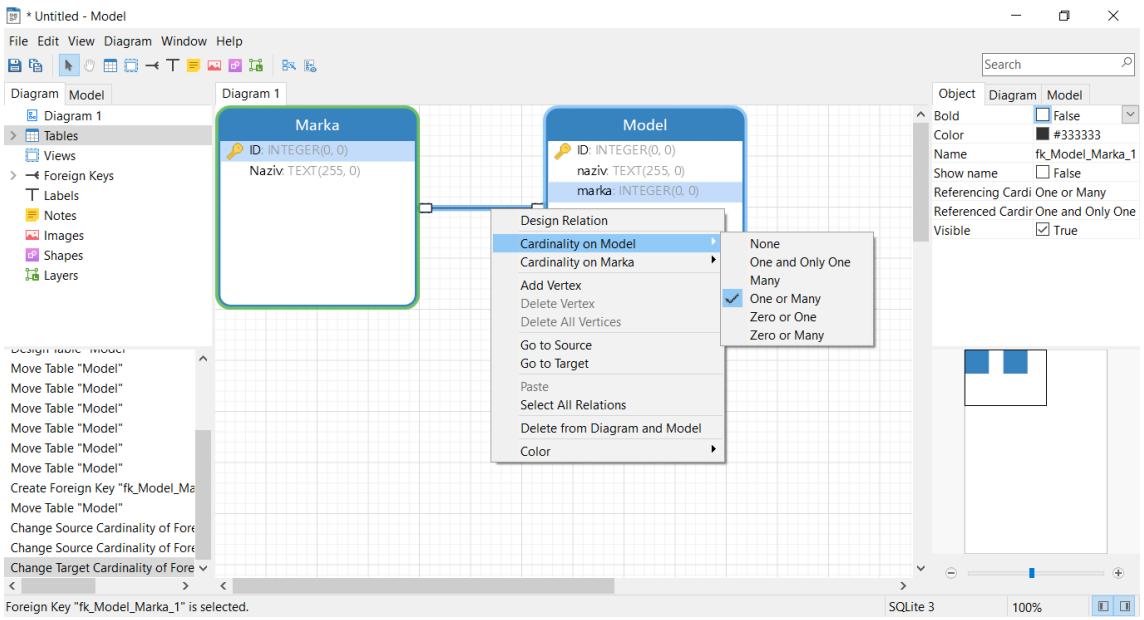
[2] Primarni ključ – ograničenje nad stupcem koje ograničava vrijednost stupca (Rabuzin K., 2011, str. 28).

Tako izgleda grafičko kreiranje tablice, koje je vrlo jednostavno i intuitivno, no iza toga stoji SQL kod koji se prilikom kreiranja tablica sam generira. Za gore kreiranu tablicu generirana SQL naredba izgleda ovako:

```
CREATE TABLE "Marka" (
      "ID" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
      "Naziv" TEXT(255) NOT NULL );
```

Na taj način se mogu kreirati sve tablice, a da bi model bio koristan, potrebno je kreirane tablice međusobno logički povezati. Povezivanje se vrši pomoću vanjskih ključeva[3] (engl. *foreign keys*). U jednoj tablici je glavni, primarni ključ, a u drugoj tablici je atribut istog tipa podatka kao i primarni ključ koji zovemo vanjski ključ te se pomoću njega povezujemo na primarni ključ.

U Navicatu se dvije tablice povezuju na način da se odabere ikona *Foreign keys,* koja je u obliku vranine noge (engl. *Crow's foot*). Zatim se odabere, odnosno lijevim klikom miša povuče vanjski ključ prema primarnom ključu, te se na taj način ostvari veza među dvije tablice. Najbolje je to predočeno na slici ispod (*Slika 6*).

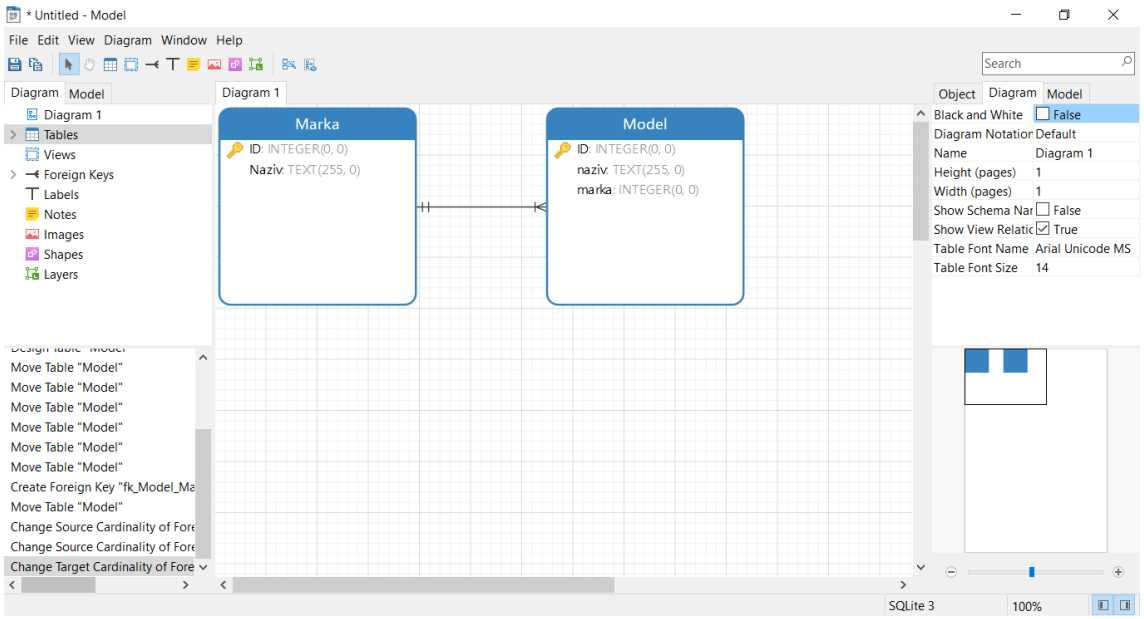


Slika 6. Veza između dviju tablica (vlastita izrada)

Nakon što se uspješno povežu dvije tablice potrebno je odrediti vrstu veze, odnosno kardinalnost (engl. *cardinality*) i opcionalnost[4]. Postoje više vrsta veza, gdje na svakoj strani tablice možemo imati sljedeće, nula ili jedan, samo jedan, nula ili više, jedan ili više i više. Ovisno kakve su potrebe realnog sustava takva se i kardinalnost postavi.

[3] Vanjski ključ – stupac u drugoj tablici u kojoj pohranjujemo vrijednost primarnog ključa (Rabuzin K., 2011, str. 29).
[4] Kardinalnost i opcionalnost – minimalni broj sudionika u vezi predstavlja opcionalnost, a maksimalni kardinalnost (Rabuzin K., 2014, str. 98).

U Navicatu, se kardinalnost određuje tako da se odabere veza te se na nju pritisne desnim klikom miša, zatim se otvori mali izbornik koji nam nudi unos kardinalnosti za tablice s oba kraja veze. Navedena akcija je prikazana na *Slika 6,* a u scenariju na *Slika 7* prikazana je veza jedan naprema više (1:M), što znači da se Marka pojavi u tablici Model jednom ili više puta, a jedan Model sadrži samo jednu Marku.



Slika 7. Kardinalna veza (vlastita izrada)

Također iza takve vrste veze stoji SQL naredba pomoću kojeg se i kreiraju sve kasnije tablice. SQL naredbe za tablice iznad je prikazan u isječku ispod. Također je prikazan isječak za tablicu s dvokomponentnim[5] primarnim ključem gdje su oba ključa ujedno i vanjski ključevi.

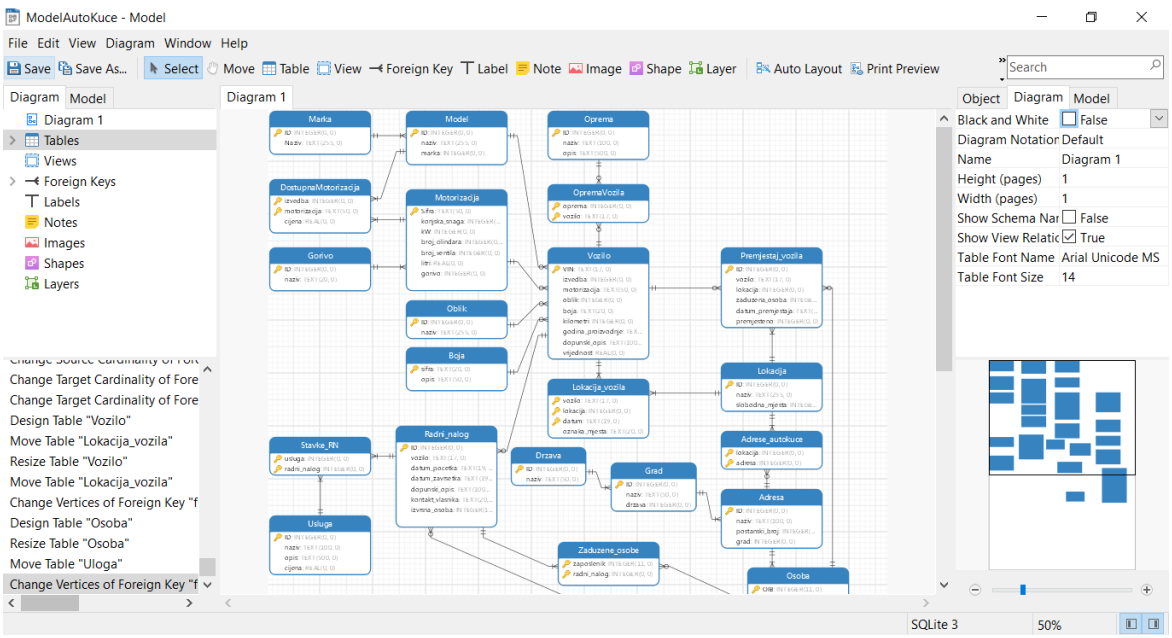
```
CREATE TABLE "Model" (
      "ID" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
      "naziv" TEXT(255) NOT NULL,
      "marka" INTEGER NOT NULL,
      CONSTRAINT "fk_Model_Marka_1" FOREIGN KEY ("marka") REFERENCES
      "Marka" ("ID") );

CREATE TABLE "DostupnaMotorizacija" (
      "izvedba" INTEGER NOT NULL,
      "motorizacija" TEXT(50) NOT NULL,
      "cijena" REAL,
      PRIMARY KEY ("izvedba", "motorizacija") ,
      CONSTRAINT "fk_DostupnaMotorizacija_Model_1" FOREIGN KEY
      ("izvedba") REFERENCES "Model" ("ID"),
      CONSTRAINT "fk_DostupnaMotorizacija_Motorizacija_1" FOREIGN KEY
      ("motorizacija") REFERENCES "Motorizacija" ("Sifra") );
```

[5] Dvokomponentni primarni ključ – ključ kojeg čine dva stupca, a njihova kombinacija (vrijednost) mora biti jedinstvena

Na takav se način izrade i povežu sve tablice unutar sustava čime se dobije konačni model (*Slika 8*). Gotovo sličan način izrade baze podataka se koristi u drugim grafičkim alatima uz neke male razlike, ali sami rezultat je uvijek isti.

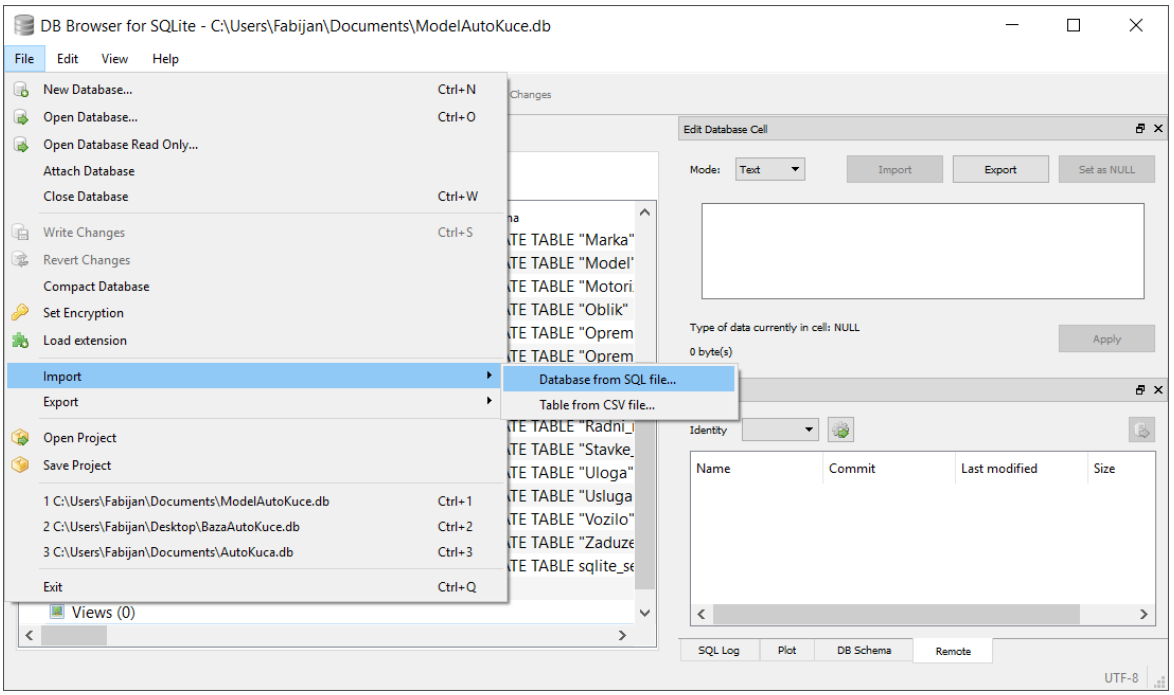
3.2. Stvaranje konačne baze podataka

Model je potrebno spremiti kako bi se mogao kasnije po potrebi modificirati, te se sad može taj isti model sinkronizirati s bazom podataka. Sinkronizacija se vrši na način da se odabere sljedeće, *File/Syncronise to Database/Sync with selected schemas*, zatim se odabere željena konekcija (engl. *Target Connection*) te željena shema, nakon toga se pritisne *Next/Compare* i onda se izgeneriraju SQL naredbe za kreiranje tablica. Odaberu se željene naredbe i pritisne se *Run Query*. Na taj način se dobije baza podataka, odnosno datoteka s ekstenzijom .db koja je prenosiva te se može iskoristiti na odgovarajućem mjestu.



Slika 8. Konačni model baze podataka (vlastita izrada)

Drugi način, ali isto vrlo sličan, je taj da se koristi alat DB Browser for SQLite, a kod modela je potrebno sljedeće odabrati, *File/Export SQL* gdje se odabere lokacija spremanja SQL skripte, odaberu se tablice i/ili pogledi koji se žele koristiti, a zatim se pritisne *Ok.* Nakon toga je potrebno otvoriti alat DB Browser for SQLite u kojem se odabere *File/Import/Database from SQL file* te se pronađe generirana SQL skripta koja se odabere, pritisne se Otvori (engl. *Open*) te se odabere mjesto spremanja .db datoteke i naziv iste, a zatim se pritisne *Ok,* čime se dobije .db datoteka koja se može koristiti i po potrebi prenositi. Opisani koraci su prikazani na slici ispod (*Slika 9*), a spomenuti alat se može koristiti za pregledavanje podataka unutar same baze podataka.



Slika 9. DB Browser for SQLite (vlastita izrada)

3.3. Kreiranje okidača

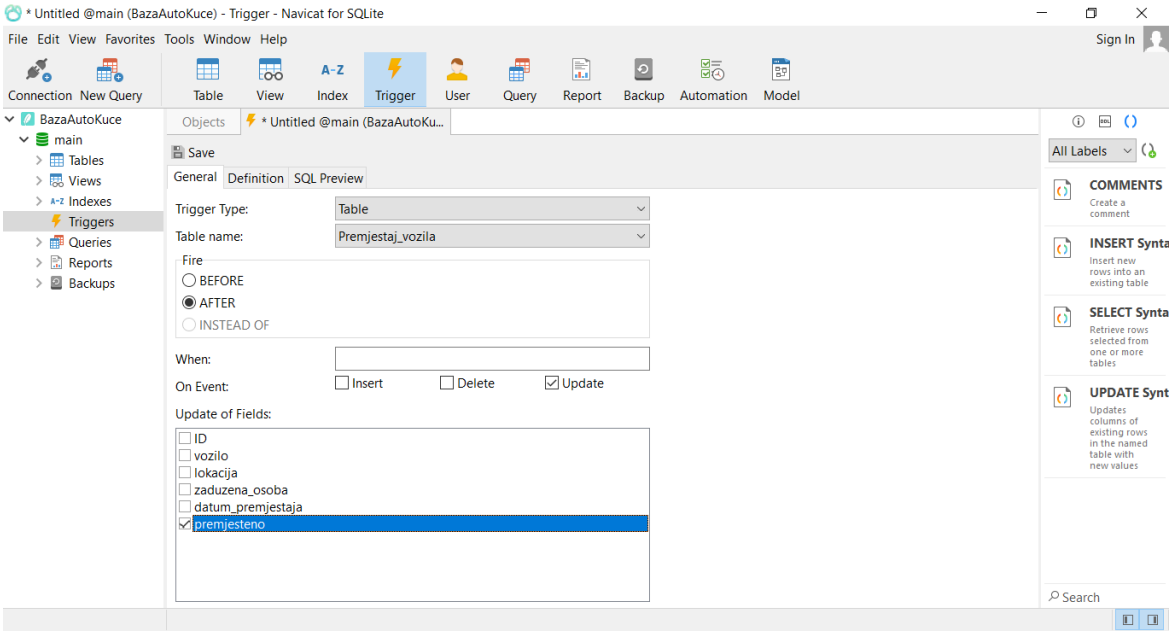
Okidač[6] se može direktno pisati unutar generirane SQL skripte, no tada je teško testirati radi li okidač ispravno ili ne. Radi toga će kreiranje okidača biti prikazano u Navicat alatu, a za pisanje okidača je potrebna veza (konekcija) na bazu podataka koja je opisana u prethodnom dijelu.

Kreiranje okidača u SQLite-u je dosta ograničeno, jer nisu podržane nikakve logičke operacije tipa *if, for, while,* itd., već su okidači primarno zamišljeni za osnovne naredbe čime bi se rad baze podataka automatizirao ili olakšao.

U Navicat alatu se okidač kreira na način da se odabere ikona ispod koje piše *Trigger*, a zatim *New Trigger* (*Slika 10*). Pod *General* se otvori sučelje za kreiranje okidača, gdje je potrebno odabrati tablicu nad kojom okidač reagira, je li vrijeme aktiviranja prije ili poslije (engl. *before, after*) početka izvršenja neke SQL naredbe. Nakon toga se specificira uvjet u kućici *when* u koju će unos biti omogućen samo ako je označen događaj *Update,* u tom slučaju se u tu kućicu upisuje logički uvjet koji treba biti ispunjen kako bi se okidač izvršio (primjer korištenja *when* naredbe je okidač „Unos nove lokacije vozila nakon premjestaja“). Kao zadnje, potrebno je definirati na koji događaj (engl. *event*) reagira, odnosno hoće li se aktivirati kod izvršavanja *Insert, Update* ili *Delete* naredbe.

[6] Okidač - automatsko izvršavanje određenih akcija kao reakcija na događaje koji se mogu dogoditi i ili pak van same baze podataka (Rabuzin K., 2014, str. 57).

U slučaju da se odabere *Update,* onda se može specificirati polje nad kojim okidač reagira, a to je ovisno od tablice do tablice. Nadalje, potrebno je otići pod karticu *Definition,* gdje se piše sami okidač, odnosno logika. Za to koristimo varijable *NEW i OLD* pomoću kojih pristupamo podacima prije i poslije izvršenja okidača. Nad događajem *Insert* dostupna je samo *NEW* varijabla, nad *Update* je *NEW i OLD,* a nad *Delete* je dostupna samo varijabla *OLD.*



Slika 10. Kreiranje okidača (vlastita izrada)

U nastavku će biti prikazana četiri okidača koja su korištena u projektu, gdje svaki ima svoju svrhu. Kod okidača su korištene razne kombinacije okidanja samog okidača (*Before* i *After*) i događaja kako bi se dobila potpuna slika korištenja okidača unutar SQLite baze podataka.

Prvi okidač je „Brisanje dostupnih motorizacija po određenoj izvedbi“ koji prilikom brisanja nekog modela automatski briše i sve Dostupne motorizacije u kojima se nalazi brisani model. Na taj način je izbjegnuta greška, narušavanje referencijalnog integriteta, kod tablice „DostupnaMotorizacija“, jer se u njoj nalazi dvokomponentni primarni ključ, a brisanje vrijednosti jednog od ključeva nije moguće, pa je prvo potrebno izbrisati potreban red iz tablice „DostupnaMotorizacija“, a tek onda iz „Model“.

```
CREATE TRIGGER "main"."Brisanje dostupnih motorizacija po određenoj
izvedbi"
BEFORE DELETE ON "Model"
BEGIN
      DELETE FROM DostupnaMotorizacija WHERE izvedba=OLD.ID;
END;
```

Sljedeći okidač je „Provjera slobodnih mjesta prilikom unosa premjestaja vozila“. Radi se o okidaču koji prilikom unosa „Premjestaj_vozila“ za unesenu lokaciju provjerava postoje li na toj lokaciji slobodna mjesta ili ne. Ako je lokacija popunjena, okidač prekida unos i ispisuje grešku s porukom čime je onemogućen unos premještaja. Greška se stvara s naredbom Raise.

```
CREATE TRIGGER "main"."Provjera slobodnih mjesta prilikom unosa
premjestaja vozila" BEFORE INSERT ON "Premjestaj_vozila"
BEGIN
SELECT CASE
      WHEN (SELECT slobodna_mjesta FROM Lokacija WHERE
      ID=NEW.lokacija)==0
      THEN RAISE(ABORT,'Korisnikova greška - Na lokaciji nema više
      slobodnih mjesta!')END;END;
```

Idući okidač je „Unos nove lokacije vozila nakon premjestaja“, a navedeni okidač služi za automatski unos podataka u tablicu „Lokacija_vozila“ ako je premještaj iz tablice „Premjestaj_vozila“ obavljen, odnosno ako je u stupcu premjesteno vrijednost prebačena sa 0 na 1, budući da nema *bool* tipa podatka. U okidaču možemo vidjeti da je korištena naredba *when* u koju je upisana logička naredba, a ako je naredba istinita onda se izvode naredbe u nastavku okidača. Za unos datuma premještaja se koristi SQL funkcija DATETIME ('now') koja uzima trenutni sustavski datum prilikom okidanja okidača.

```
CREATE TRIGGER "main"."Unos nove lokacije vozila nakon premjestaja"
AFTER UPDATE OF "premjesteno" ON "Premjestaj_vozila" WHEN
NEW.premjesteno!=0
BEGIN
  INSERT INTO Lokacija_vozila (vozilo,datum,lokacija)
  VALUES(OLD.vozilo,DATETIME('now'),OLD.lokacija);
END;
```

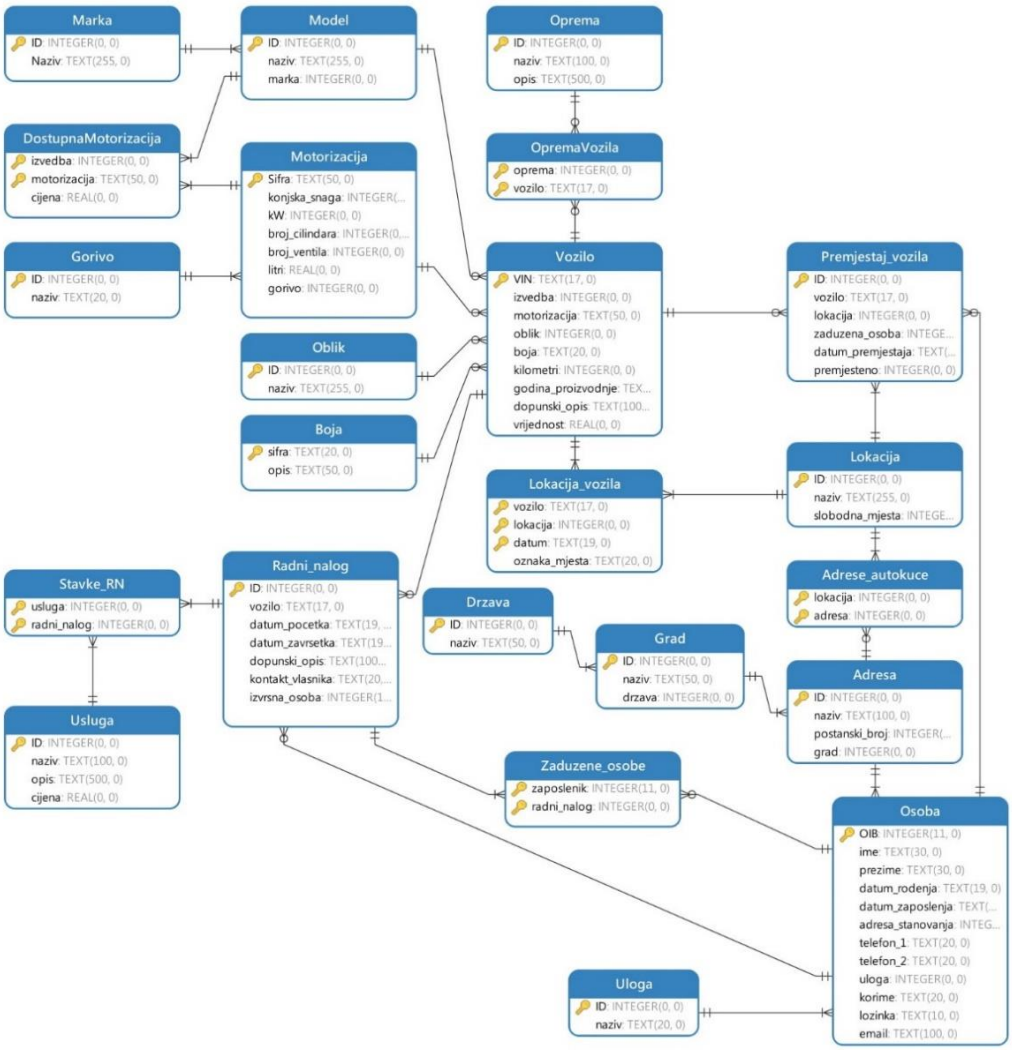
Zadnji okidač je „Izmjena broja slobodnih mjesta na lokaciji“, koji služi tome da se broj slobodnih mjesta umanji za 1 na određenoj Lokaciji i to tada kada je premještaj obavljen. Ovako kako je sada okidač napisan izgleda da broj mjesta može otići i u minus područje, no to nije moguće zbog okidača „Provjera slobodnih mjesta prilikom unosa premjestaja vozila“ koji će zabraniti unos premještaja ako je broj mjesta jednak nuli.

```
CREATE TRIGGER "main"."Izmjena broja slobodnih mjesta na lokaciji"
BEFORE INSERT ON "Lokacija_vozila"
BEGIN
  UPDATE Lokacija SET slobodna_mjesta=slobodna_mjesta-1 WHERE
  ID=NEW.lokacija;
END;
```

4. Implementacija aplikacije za upravljanje auto kućom

Dosad je izrađena baza podataka, ali je potrebno tu istu bazu podataka prikazati u grafičkom obliku kako bi se prilikom kasnije implementacije programer ili neka druga osoba mogla lakše snalaziti. U te svrhe se izrađuje ERA model[7] (entitet relacija atribut) koji na grafički način prikazuje tablice i njene atribute te način na koje su one povezane, koji su njihovi međusobni odnosi, odnosno koja je struktura baze podataka. ERA model se može prikazati na više načina, također se može izraditi u više alata, a budući da je baza podataka kreirana unutar grafičkog sučelja Navicat lako je iz njega izvući napravljeni model.

4.1. ERA model baze podataka



Slika 11. ERA model auto kuće (vlastita izrada)

[7] ERA model- vrsta dijagrama koji omogućuje prikaz korisničkih zahtjeva, odnosno definiranje strukture baze (Rabuzin K., 2014, str. 95).

Iznad prikazani ERA model (*Slika 11*) promatramo putem entiteta[8], atributa[9] i veze[10] gdje je entitet tablica, a atribut opis tablice. Atributi, a time i tablice, su povezani pomoću zasebnih veza čime se tvori struktura modela. U samom modelu postoje 23 tablice, tj. entiteta, a od 23 tablice njih 5 su vezivne (asocijativne), odnosno slabi entiteti, koji sadrže dvokomponentni primarni ključ pomoću kojih se povezuju dvije tablice.

Prema ERA modelu u poslovnom svijetu auto kuće postoji entitet „Marka“, a time i tablica „Marka“. Tablici je dodijeljen atribut „ID“ koji je ujedno primarni ključ, te atribut „naziv“. Atribut „ID“ je podvučen jer je primarni ključ te je on identifikator za tablicu „Marka“, a „naziv“ nije te njega zovemo deskriptorom. Iduća tablica je tablica „Model“, koja također ima identifikator „ID“, deskriptor „naziv“ te vanjski ključ „marka“ koji se veže na tablicu „Marka“ i time se ostvaruje veza jedan naprema jedan ili više (1:M).

Iduća tablica je „Motorizacija“, koja sadrži nekoliko atributa. Prvi atribut i ujedno identifikator je „Sifra“, deskriptori su „konjska_snaga“, „kW“, „broj_cilindara“, „broj_ventila“, „litri“ gdje svi sadrže brojčanu vrijednost. Tablica „Motorizacija“ sadrži jedan vanjski ključ, „gorivo“, koji se veže na tablicu „Gorivo“. Tablica „Gorivo“ sadrži identifikator „ID“ te deskriptor „naziv“, a ostvarena veza između „Motorizacija“ i „Gorivo“ je tipa jedan naprema jedan ili više (1:M).

Tablica „DostupnaMotorizacija“ je jedna od vezivnih tablica (slabi entitet), te ona služi tome da se nekom modelu dodijeli motorizacija, no svaki model može imati više motorizacija (npr. Mercedes-Benz C200, C220, C250), te iz tog razloga postoji vezivna tablica, kojom se implementira veza više naprema više (N:M). Tablica sadrži dva identifikatora, koji su ujedno i vanjski ključevi, „izvedba“, „motorizacija“, te deskriptor „cijena“ koji označava osnovnu cijenu koja je bila postavljena za vozilo prilikom prodaje, ali je nije potrebno unijeti. Veza do „Motorizacije“ i „Modela“ je tipa jedan naprema jedan ili više (1:M).

Tablica „Oblik“ sadrži sve oblike vozila, tipa karavan, limuzina, monovolumen, itd. Atributi su sljedeći, identifikator „ID“ te deskriptor „naziv“. Sljedeća tablica je „Boja“, koja za identifikatora sadrži „sifra“ te „opis“ u koji se može upisati naziv boje ili opis, ako ne postoji potpuni naziv iste.

[8] Entitet – objekt o kojem se prikupljaju informacije (Rabuzin K., 2014, str. 95).
[9] Atribut – svojstva entiteta koja se žele pohraniti u bazi podataka (Rabuzin K., 2014, str. 96).
[10] Veza – ispravno nazvano tip veze, imenovana asocijacija između tipova entiteta (Rabuzin K., 2014, str. 96).

Zadnja tablica koja čini jedno vozilo, je „Oprema“. Onda sadrži identifikator „ID“ te deskriptore „naziv“ i „opis“. „Oprema“ se veže s tablicom „Vozilo“ putem vezivne tablice „OpremaVozila“ kako bi se izbjegla veza više naprema više (N:M). „OpremaVozila“ sadrži 2 primarna ključa koji su ujedno i primarni, a to su „oprema“ i „vozilo“. Tako ostvarena veza je tipa jedan naprema nula ili više s obje strane (1:M).

Iduća tablica je tablica „Vozilo“, a ona sadrži identifikator „VIN“, deskriptore „kilometri“, „godina_proizvodnje“, „dopunski_opis“, „vrijednost“. Ostali atributi su vanjski ključevi, „izvedba“, „motorizacija“, „oblik“ i „boja“ koji se vežu na istoimene tablice, osim „izvedbe“, ona se veže na „Model“. Sve veze vanjskih ključeva su tipa jedan naprema nula ili više (1:M), npr. vozilo se obavezno sastoji od marke i modela, ali model u sklopu auto kuće ne treba biti zauzet od nekog vozila.

Vozila se mogu premještati unutar površina auto kuće, tj. s lokacije na lokaciju. Zahtjev za novim premještajem se evidentira unutar tablice „Premjestaj_vozila“, gdje je identifikator „ID“, a deskriptori su „datum_premjestaja“ i „premjesteno“. Ostali atributi su vanjski ključevi, „vozilo“, „lokacija“ i „zaduzena_osoba“. Veza od „Vozila“ do „Premjestaj_vozila“ je tipa jedan naprema nula ili više (1:M) jer se neko vozilo možda ne treba nikada premjestiti, a neko drugo možda više puta. Veza od „Premjestaj_vozila“ do „Lokacije“ je jedan naprema jedan ili više (1:M), a zadnja veza od „Premjestaj_vozila“ do „Osoba“ je jedan naprema nula ili više (1:M), gdje neka osoba radi u drugom odjelu pa nikada neće trebati obaviti funkciju premještaja vozila.

Tablica „Lokacija“ sadrži identifikator „ID“ i deskriptore „naziv“ i „slobodna_mjesta“, gdje se lokacija odnosi na lokaciju auto kuće, a jedna auto kuća može imati više lokacija. Iduća tablica je „Lokacija_vozila“, gdje imamo tri primarna ključa, „vozilo“, „lokacija“ i „datum“, a razlog tome je da se jedno vozilo može nalaziti samo na jednom mjestu, lokaciji unutar nekog razdoblja, zato je i datum ovdje primarni ključ. Jedini deskriptor je „oznaka_mjesta“, u koji unosi broj ili oznaka mjesta na kojem se vozilo nalazi.

Iduće tablice služe za prikaz adresa osoba, lokacija, smještaja i slično. Tablica „Drzava“ sadrži identifikator „ID“ i deskriptor „naziv“. Tablica „Grad“ sadrži identifikator „ID“ i deskriptor „naziv“ te vanjski ključ „drzava“, čime se ostvaruje veza jedan naprema jedan ili više (1:M). Zadnja takva tablica je „Adresa“ gdje imamo identifikator „ID“, deskriptore „naziv“ i „postanski_broj“ i vanjski ključ „grad“ gdje se također ostvaruje veza jedan naprema jedna ili više (1:M) do tablice „Grad“.

„Adresa“ je povezana s „Lokacijom“ putem vezivne tablice (slabog entiteta) „Adrese_autokuce“, budući da jedna lokacija auto kuće može imati više adresa, npr. drugi dio parkirališta jedne lokacije je na drugoj adresi, ali i dalje u blizini npr. zbog manjka prostora. Takve dvije adrese jedne auto kuće se interno vode pod jednom lokacijom. Tako imamo dva vanjska ključa, „lokacija“ i „adresa“, koji su povezani na istoimene tablice vezom jedan naprema jedan ili više (1:M) i jedan naprema nula ili više (0:M).

Nadalje imamo, tablicu „Osoba“, u kojoj su zapisani zaposlenici, ali mogu se unositi i kupci. Tako imamo identifikator „OIB“, deskriptore „ime“, „prezime“, „datum_rodenja“, „datum_zaposlenja“, „telefon_1“, „telefon_2“, „korime“, „lozinka“ i „email“, te vanjske ključeve „adresa_stanovanja“ i „uloga“. Oba vanjska ključa su povezani s vezom jedan naprema jedan ili više (1:M) . Osoba ima samo jednu adresu i ulogu, ali se adresa može pojaviti više puta za osobe, npr. bračni par, dok osoba ima samo jednu ulogu unutar poduzeća npr. menadžer, mehaničar, prodavač.

Tablica „Usluga“ sadrži sve poslovne aspekte koje nudi jedna auto kuća, popravak, servis, pranje i slično. Njezini atributi su identifikator „ID“, deskriptori „naziv, „opis“ i „cijena“. „Usluga“ se veže na „Radni_nalog“ putem slabog entiteta „Stavke_RN“, koja sadrži vanjske primarne ključeve, „usluga“ i „radni nalog“, dok „Radni_nalog“ sadrži sljedeće identifikator „ID“ te deskriptore „datum_pocetka“, „datum_zavrsetka“, „dopunski_opis“ i „kontakt_vlasnika“. Ostali atributi su ustvari vanjski ključevi, „izvrsna_osoba“ koji se veže na „Osoba“, te „vozilo“ koji se veže na „Vozilo“. Obje veze su jedan naprema nula ili više (1:M) iz razloga što nije svaka osoba zadužena za usluge auto kuće (npr. računovodstvo), a niti ne treba svako vozilo biti izvršni objekt neke usluge, već može biti samo na prodaji.

Zadnja tablica, odnosno entitet je „Zaduzene_osobe“. Navedena tablica postoji iz razloga što na jednom radnom nalogu može raditi više osoba, čime se stvara veza više naprema više (N:M), a takvi slučajevi se izbjegavaju uvođenjem slabog entiteta. Atributi navedene tablice su primarni vanjski ključevi „zaposlenik“ i „radni nalog“, gdje je veza na „Osobu“ jedan naprema nula ili više (1:M), a na „Radni_nalog“ jedan naprema jedan ili više (1:M).

Time su opisani svi entiteti i njihovi atributi, te odnosi između pojedinih entiteta. Model se mogao proširiti s uvođenjem nekoliko entiteta, gdje bi neki bili zaduženi za potpuni opis usluga koje se nude, jer ovako svaka usluga ima fiksnu cijenu za bilo koje vozilo, a u realnosti nije to tako, već neka vozila zahtijevaju više posla od drugih pa je i cijena također veća.

4.2. Opis razvojne okoline

Svaka izrađena baza podataka se na neki način mora i koristiti, a u svrhe ovog rada će se izraditi mobilna aplikacija s grafičkim sučeljem, koje će biti način komuniciranja osobe sa samom bazom podataka. Primjer baze podataka (auto kuće) i mobilne aplikacije nije potpuno realan, već on služi tome da se prikažu sve pojedinosti SQLite sustava za upravljanje bazom podataka i načina povezivanja i rada s njom.

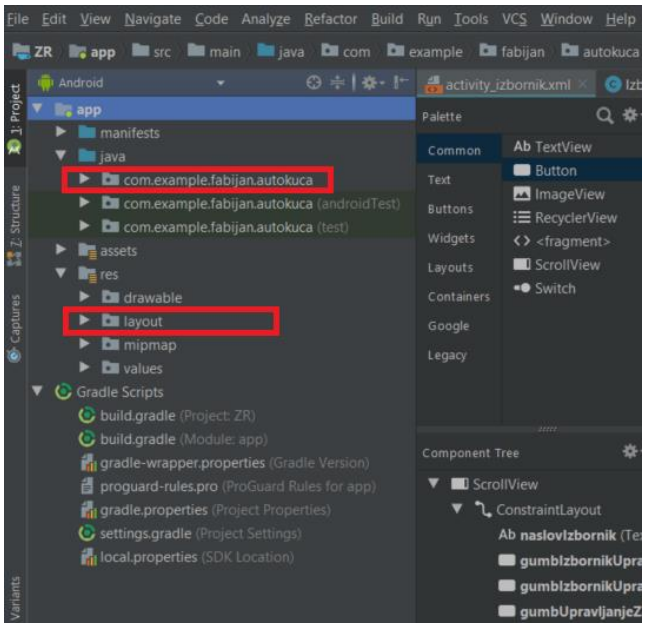
Budući da se radi o mobilnoj aplikaciji, interakcija s njom je zamišljena putem dodira grafičkih elemenata na zaslonu. Moći će se unositi novi podaci, poput vozila, lokacija vozila i slično, brisati i izmijeniti podaci itd.. Korisnik neće trebati imati nikakvog znanja o sintaksi i naredbama za upravljanje bazom podataka, već je tu aplikacija i cjelokupno sučelje koje korisniku olakšava sam rad s podacima.

Sama aplikacija će biti izrađena u razvojnom okruženju Android Studio u kojem se koristi Java programski jezik, što je i *de facto* standard na tržištu mobilnih aplikacija, bar što se tiče Android uređaja. Android Studio je razvojno okruženje za izradu Android aplikacija, a baziran je na IntelliJ IDEA-u, sučelju za razvoj aplikacija u Java programskom jeziku (Developers, 2018.). Na drugoj strani je Java programski jezik koji je na tržište 1995. od strane Sun Microsystema te se koristi u svim mogućim područjima elektronike (računala, mobilnih uređaja, konzola) (Oracle, 2018.).

4.2.1. Uvod u Android Studio

Java je zadužena za pozadinski dio izvršavanja koda, no treba i omogućiti interakciju koda s grafičkim sučeljem, a zato je tu Android Studio. Prvo i najbitnije je da se zna gdje se nalazi programski dio koda kojim se kasnije manipulira. Unutar samog projekta se nalaze nekoliko mapa, a za izradu aplikacije najbitnije su app/java/nazivProjekta i app/res/layout koje su prikazane na idućoj slici (*Slika 12*).

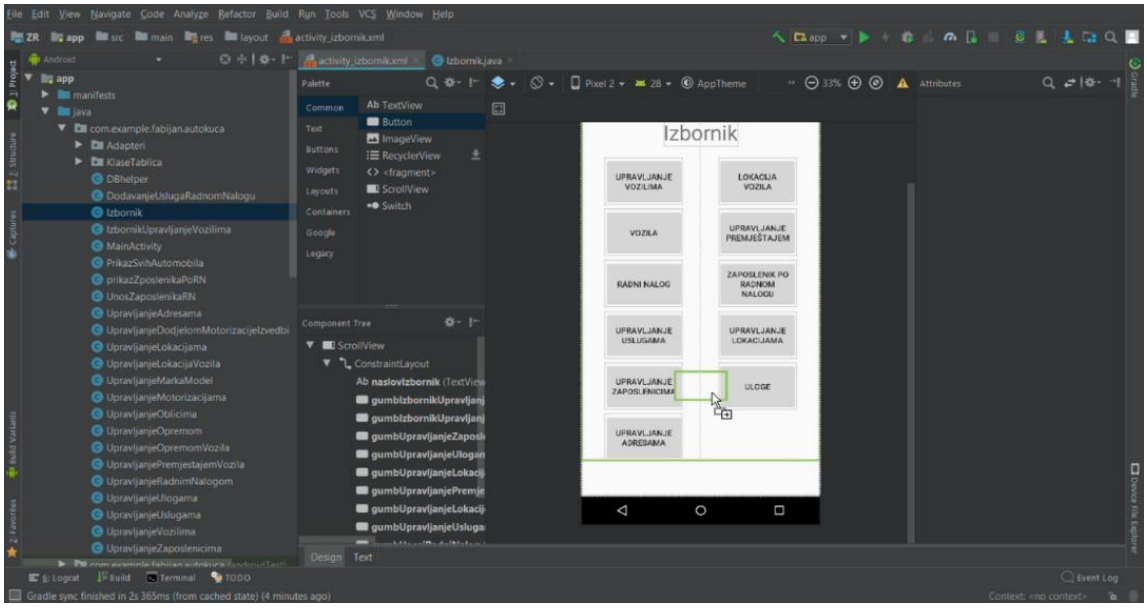
U prvoj mapi se nalaze klase koje sadrže glavni dio koda, tj. ono što bi aplikacija trebala raditi, a u drugoj mapi se nalaze .xml datoteke koje sadrže kod pomoću kojeg se upravlja izgledom zaslona, animacija na zaslonu i slično. Dijelom izgleda zaslona se može manipulirati iz samih klasa, pomoću raznih metoda, no ako nema potrebe za tim, tada takav način treba izbjegavati.



Slika 12. Organizacija mape projekta (vlastita izrada)

Za početnike je vrlo jednostavno pohvatati one najkorištenije i one najbitnije stvari, uglavnom se rad s Android Studiom bazira na „*drag and drop*“ principu (*Slika 13*). Odabere se neki element, tipa gumb, kućice za upis podataka ili padajući izbornik koji se postavi na zaslon uređaja, ali prije korištenja podataka je potrebno inicijalizirati te postavljene elemente.

Takav način kreiranja sučelja je vrlo jednostavan i intuitivan ali i relativno ograničen, za finije i preciznije upravljanje elementima na zaslonu se preporučuje ručno pisanje elementa unutar .xml datoteke. Pisanjem unutar .xml datoteke mogu se izgraditi novi grafički elementi koji mogu ispuniti neke specifične zahtjeve. Također je moguće preuzeti već gotova rješenja s interneta, ako ona postoje.



Slika 13. Postavljanje elementa na zaslon (vlastita izrada)

U projektu su uglavnom korišteni sljedeći elementi, *TextView* za prikaz naslova i naziva, *EditText* za unos tekstualnih podataka, *Button* za potvrdu različitih naredbi i *Spinner* za prikaz padajućih listi. Inicijalizacija novo postavljenih elemenata unutar Jave vrši se na sljedeći način:

```
...

public class Izbornik extends AppCompatActivity {

      private Button gumb1;
      private TextView textview1;
      private EditText edittext1;
      private spinner spinner1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_izbornik);

           gumb1=(Button)findViewById(R.id.gumb1);
           textview1=(TextView)findViewById(R.id. textview1);
           edittext1=(EditText)findViewById(R.id. edittext1);
           spinner1=(Spinner)findViewById(R.id. spinner1);

      }
}
```

Zadnja bitna stvar, za izradu aplikacije je korišten Android 5.0 verzija (Lollipop) tj. API 28 verzija, što bi trebalo odgovarati velikoj većini današnjih Android uređaja, a kao testni uređaj je korišten Google Pixel 2.

4.2.2.Uvod u Java programski jezik

Kao i svaki programski jezik četvrte generacije (objektno orijentirano programiranje) tako i ovdje imamo četiri gradivna elementa: Objekt, Klasa, Metoda i Varijable. Bitno je znati da se svaki programski kod nalazi unutar nekog dijela bloka, a sam blok naredbi se označava s znakovima „{}“, a druga bitna stvar je ta da svaka naredba završava sa znakom „;“ koji ustvari terminira naredbu. Bitno je i da se zna što su modifikatori pristupa[11], a oni su *private*, *protected*, *public*, *no modifier*. Oni obilježavaju „dostupnost“ pojedinih elemenata klase, tako npr. ako je klasa označena s *private*, ne možemo joj pristupiti iz neke druge klase.

[11] Modifikatori pristupa – svojstvo koje određuje do koje razine je napisani kod dostupan (Radošević D., 2007, str. 222).

```
public class Izbornik extends AppCompatActivity {

    private int broj1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_izbornik);

        broj1=5;
     }

     private void NekaMetoda()
     {
          int broj2=10;
     }
}
```

Budući da je aplikacija namijenjena za mobilne uređaje koji se koriste s rukama, odnosno svaka interakcija se odvija putem dodira ekrana potrebne su i funkcije koje mogu reagirati na te naredbe, njih zovemo događaji (engl. *events*). Tako npr. za gumb se često koristi funkcionalnost .*setOnClickListener*, događaj koja se u radu često koristi kako bi se nešto potvrdilo ili kako bi se prešlo na idući, novi, zaslon.

```
gumb.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View v) {
          programski kod…
     }
});
```

Također je moguće vršiti razne interakcije s unosom teksta pomoću .*addTextChangedListener*, gdje je moguće preuzeti upisan tekst prije upisivanja novih znakova, u trenutku upisa znaka i nakon upisa novih znakova. Takva funkcionalnost je korisna prilikom upisa riječi koju je potrebno pronaći u bazi podataka (filtriranja).

```
edittext.addTextChangedListener(new TextWatcher() {
     @Override
     public void beforeTextChanged(CharSequence s, int start, int
     count, int after) {
          ...
     }

     @Override
     public void onTextChanged(CharSequence s, int start, int
     before, int count) {
```

```
            ...
        }

        @Override
        public void afterTextChanged(Editable s) {
            ...
        }
});
```

Java programski jezik također omogućuje pisanje opće poznatih petlji kao što su *while*, *do while*, *for*, koje omogućuju ponavljanje nekog dijela koda iz nekoliko razloga. Također postoje naredbe za grananje kao što su *if* i *switch case*.

4.3. Objekti tablica

Kako bi se pojednostavio rad s podacima koji se upisuju i ispisuju iz baze podataka koristi se jedna značajka 4. generacije programskih jezika, a to su Objekti[12]. Sami objekti u projektu se nalaze u mapi app/ java/ nazivProjekta/ KlaseTablica.

U ovome slučaju su svi objekti reprezentacija jedne od tablica, svi atributi koji se nalaze u tablici nalaze se i u samom objektu. Varijable također mogu biti i drugi objekti, tako da u slučaju pojavljivanja vanjskog ključa u jednoj tablici, u sami objekt se postavlja varijabla tipa nekog objekta na koji se veže originalan objekt. Najbolje je to opisano i prikazano programskim odsječkom u nastavku.

Prvi objekt je objekt tablice „Usluga“, koji ne sadrži niti jedan vanjski ključ. U klasi postoje setteri i getteri[13] pomoću kojih se vrši unos podataka u klasu te izvlačenje podataka iz klase. Također je bitno napomenuti metodu „toString“ s anotacijom Override pomoću koje se stvara odgovarajući prilagođeni ispis podataka.

```
public class Usluga {

    private int ID;
    private String naziv;
    private String opis;
    private float cijena;

    public int getID() {
        return ID;
    }
```

[12] Objekt – kolekcija operacija koje dijele zajedničke operacije, omogućuje trajne usluge kroz vrijeme (Radošević D., 2007, str. 111).
[13] Metode pomoću kojih se rade izmjene nad varijablama (dohvaćanje i postavljanje vrijednosti varijable)

```
    public void setID(int ID) {
        this.ID = ID;
    }

    ...

    @Override
    public String toString() {
        return this.naziv+" "+this.cijena+"kn: "+this.opis;
    }
}
```

Idući objekt je objekt tablice „Vozilo“, tablica koja se većim dijelom sastoji od vanjskih ključeva te time sadrži veliki broj varijabli.

```
public class Vozilo {
    private String VIN;
    private int izvedba;
    private String motorizacija;
    private int oblik;
    private String boja;
    private int kilometri;
    private String godina_proizvodnje;
    private String dopunski_opis;
    private float vrijednost;
    private Model Mod;
    private Motorizacija Mot;
    private Oblik Obl;
    private Boja Boj;
    private Marka mark;

    public String getVIN() {
        return VIN;
    }

    public void setVIN(String VIN) {
        this.VIN = VIN;
    }

    public int getIzvedba() {
        return izvedba;
    }

    public void setIzvedba(int izvedba) {
        this.izvedba = izvedba;
    }
      ...
```

```
    public Model getMod() {
        return Mod;
    }

    public void setMod(Model mod) {
        Mod = mod;
    }

    @Override
    public String toString() {
           return this.getVIN()+": "+this.getMark().getNaziv()+"
           "+this.getMod().getNaziv()+" "+this.getObl().getNaziv()+"
           "+this.getBoj().getOpis();
    }
}
```

Kao i prethodna dva objekta, na isti način je kreirano preostalih 17 objekata iz projekta, gdje je jedina razlika u tome kako je korištena metoda „toString()“. Negdje je potrebno iskoristiti sve podatke iz objekta, a negdje samo dio kako bi se stvorio odgovarajući ispis podataka.

4.4. Rad s bazom podataka

Kako bi se uopće omogućio rad s SQLite bazom podataka u Java okruženu, potrebne su dvije biblioteke, SQLiteDatabase i SQLiteOpenHelper. Prva biblioteka omogućuje korištenje metoda za manipulacijom nad bazom podataka. Metode su sljedeće: *.execSQL*, *.insert*, *.update*, *.rawQuery* i još nekoliko drugih (Developers, 2018.). Druga biblioteka upravlja kreiranjem tablica u bazi podataka te krajnjim upravljanjem nad bazom podataka. Dvije glavne podklase navedene biblioteke su *OnCreate()* i *OnUpgrade()*, a za pristup bazi podataka se koriste metode *getReadableDatabase()* i *getWritableDatabase()* (Developers, 2018.).

Rad i upravljanje nad bazom podataka je podijeljeno na dva dijela, 1. klasu u kojoj se kreira baza podataka, tj. tablice i okidači, 2. klase koji imaju sufiks „Adapter“ iz kojih se vrši konekcija na pojedine tablice, odnosno čitanje i pisanje podataka.

Klasa iz koje se kreiraju tablice se zove *DBhelper* te ona nasljeđuje svojstva i metode klase *SQLiteOpenHelper*, dok sve ostale klase (adapteri) nasljeđuju klasu DBhelper. Takva organizacija je potrebna jer bi inače sve metode za povezivanje na bazu podataka bile u jednoj klasi, što je vrlo nepregledno i dovodi to češćih pogrešaka, kao i teško ispravljanje tih pogrešaka.

Same tablice se kreiraju unutar klase *DBhelper* unutar metode *OnCreate()*, isto kao i okidači. Svaka naknadna promjena strukture baze podataka ili pojedinih tablica se vrši unutar metode *OnUpgrade().*

U nastavku u izdvojenom dijelu koda prikazana klasa DBhelper:

```
...
public class DBhelper extends SQLiteOpenHelper {
    private static int DATABASE_VERSION = 1;
    private static String DB_FILE_NAME = "AutoKuca.db";

    public DBhelper(Context context) {
        super(context, DB_FILE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

           String sql = "CREATE TABLE \"Marka\" (\n" +"\"ID\"
           INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,\n" +
           "\"naziv\" TEXT(255) NOT NULL\n" +");";
           db.execSQL(sql);

           String sql2="CREATE TABLE \"Model\" (\n" +"\"ID\" INTEGER
           PRIMARY KEY AUTOINCREMENT NOT NULL,\n"+"\"naziv\"
           TEXT(255) NOT NULL,\n" +"\"marka\" INTEGER NOT NULL,\n"
           +"CONSTRAINT \"fk_Model_Marka_1\" FOREIGN KEY (\"marka\")
           REFERENCES \"Marka\" (\"ID\")\n" +");";
           db.execSQL(sql2);
      ...

           String sql24="CREATE TRIGGER \"Brisanje dostupnih
           motorizacija po određenoj izvedbi\" BEFORE DELETE ON
           \"Model\"\n" + "BEGIN\n" + "\tDELETE FROM
           DostupnaMotorizacija WHERE izvedba=OLD.ID;\n" + "END;";
           db.execSQL(sql24);
      ...
    }

    @Override
     public void onUpgrade(SQLiteDatabase db, int oldVersion, int
     newVersion) {
          if (newVersion == oldVersion + 1) {
           //   db.execSQL("ALTER TABLE Vozilo ADD COLUMN
                broj_vlasnika INTEGER");
       }
    }
}
```

4.4.1.Unos podataka u bazu podataka

Unos podataka u bazu podataka se može obaviti na nekoliko načina, a za to postoje posebne metode iz klase *SQLiteDatabase*. U samoj aplikaciji su korištene metode *.insert* i *.rawQuery* za unos podataka. Za prvu metodu je potrebno organizirati podatke pomoću varijable *ContentValues*, gdje se svakoj vrijednosti pridružuje ključ, a ključ je u ovom slučaju naziv stupca, a kod same metode je jedino potrebno specificirati tablicu u koju se unose podaci. Još jedna bitna stvar kod metode *.insert* je ta da ona vraća -1 ako je došlo do pogreške prilikom unosa, a ako je unos bio uspješan, onda vraća identifikator (broj) novo unesenog reda koji se onda može koristiti prilikom daljnjeg rada s bazom podataka.

Primjer unosa Radnog naloga u bazu podataka putem *.insert* metode:

```
public long UnosRadnogNaloga(RadniNalog rn)
    {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues podaci=new ContentValues();
        podaci.put("vozilo",rn.getVoz().getVIN());
        podaci.put("datum_pocetka",rn.getDatum_od());
        podaci.put("datum_zavrsetka",rn.getDatum_do());
        podaci.put("dopunski_opis_rn",rn.getOpis());
        podaci.put("kontak_vlasnika_vozila",rn.getKontakt());
        podaci.put("izvrsitelj_rn",rn.getZap().getOIB());
        long id=db.insert("Radni_nalog",null,podaci);
        db.close();
        return id;
    }
```

Druga metoda je znatno jednostavnija za korištenje te zahtjeva jedino poznavanje pisanja SQL naredbi za unos podataka. Problem kod *.rawQuery* je taj da se ne može dobiti identifikator novo unesenog reda i da ne vraća nikakvu poruku ili vrijednost ako dođe do greške, što može biti problem prilikom debugiranja[14] (*eng. debugging*).

Primjer unosa Modela u bazu podataka pomoću *.execSQL* metode:

```
public void UnosModela(String naziv, Marka m)
{
      SQLiteDatabase db=this.getWritableDatabase();
      db.execSQL("INSERT INTO Model(naziv,marka) VALUES
      ('"+naziv+"','"+m.getID()+"')");
      db.close();
}
```

[14] Debagiranje - Proces pronalaženja i ispravljanja pogreška

4.4.2.Unos država i gradova

Unos država i gradova je u aplikaciji omogućen, no nema smisla svaki put unosit novu državu ili grad ako se želi unijeti nova adresa. Iz tog razloga se prilikom prvog pokretanja aplikacije podaci automatski unesu u bazu podataka.

Kako bi se to izvelo, korištena je JSON[15] datoteka u kojoj se nalaze sve prihvaćene države i pripadajući gradovi, a datoteka je smještena unutar projekta u mapi *assets*, u koju se inače smještaju druge pomoćne datoteke, slike i sl. Podaci se iz JSON datoteke čitaju uz pomoć klase *DohvatJSONdatoteke*, gdje se stvara zapis kao znakovni niz koji se kasnije koristi za čitanje.

```
public String UcitajDatoteku()
    {
        String json = null;
        try {
           InputStream is =
           context.getAssets().open("countries.json");
           int size = is.available();
           byte[] buffer = new byte[size];
           is.read(buffer);
           is.close();
           json = new String(buffer, "UTF-8");
        } catch (IOException ex) {
            ex.printStackTrace();
            return null;
        }
        return json;
    }
```

Nakon što su učitani, podaci se unose u bazu podataka, a taj dio se obavlja na prvom, početnom zaslonu i to samo prilikom prvog pokretanja aplikacije. U ovom slučaju se unose samo hrvatski gradovi jer bi inače unos podataka trajao značajno duže za sve države.

```
private void UnosDrzavaIgradova()
    {
           List<Grad> listaGardova=gradAdapter.DohvatSveGradove();

        if(listaGardova.isEmpty())
        {
            try {
                JSONObject obj = new
                JSONObject(dohvatJSONDatoteke.UcitajDatoteku());
```

[15] JSON – (JavaScript Object Notation) način pohranjivanja i organiziranja tekstualnih podataka unutar jedne datoteke

```
                JSONArray niz = obj.getJSONArray("drzave");

                for (int i = 0; i < niz.length(); i++) {

                    JSONObject object = niz.getJSONObject(i);
                     String drzava = object.getString("drzava");

                    Drzava d=new Drzava();
                    d.setNaziv(drzava);
                     int id=
                     (int)drzavaAdapter.UnosDrzave(drzava);
                    d.setID(id);

                    if(drzava.equals("Croatia")) {
                        JSONArray nizGardova =
                        object.getJSONArray("gradovi");
                for (int j = 0; j < nizGardova.length(); j++)
                     {
                     Grad g = new Grad();

                           g.setNaziv(nizGardova.getString(j));
                           g.setDrz(d);

                           gradAdapter.UnosGrada(g);
                }
                }
              }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }
```

Središnji dio koda će se tek izvršiti ako u bazi podataka u tablici „Grad“ ne postoji ni jedan zapis. Isto tako se moglo staviti i za „Drzavu“, odnosno, ako ni jedna država nije unesena, onda je potrebno popuniti tablice „Drzava“ i „Gard“.

4.4.3.Izmjena podataka

Za izmjenu podataka u bazi podataka također je moguće koristiti metodu *.execSQL* ali postoji i posebna metoda imenom *.update*. U oba slučaja je potrebno specificirati gdje se i koji podatak mijenja. Pomoću *.execSQL* metode se samo napiše standardna SQL naredba, dok je kod *.update* metode potrebno ponovo organizirati podatke isto kao što je bilo kod *.update*, te je još potrebno specificirati u kojem ili kojim se redovima izmijenjuju podaci. Metoda *.update* također vraća broj, *integer*, ovisno kako se izvela sama metoda.

Primjer izmjene podataka Zaposlenika pomoću *.execSQL* metode:

```
public void IzmjenaZposlenika(Zaposlenik z)
    {
           SQLiteDatabase db = this.getWritableDatabase();
           db.execSQL("UPDATE Osoba SET ime='"+z.getIme()+"'
           ,prezime='"+z.getPrezime()+"'
           ,datum_rodenja='"+z.getDatum_rodenja()+"'
           ,datum_zaposlenja='"+z.getDatum_zapolenja()+"'
           ,telefon1='"+z.getTelefon1()+"'
           ,telefon2='"+z.getTelefon2()+"'  " +
           "WHERE OIB="+z.getOIB());
           db.close();
    }
```

Primjer izmjene podataka Zaposlenika pomoću *.update* metode:

```
public void IzmjenaZposlenika(Zaposlenik z)
    {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues podaci=new ContentValues();
        podaci.put("ime",z.getIme();
        podaci.put("prezime",z.getPrezime());
        podaci.put("datum_rodenja",z.getDatum_rodenja());
        podaci.put("datum_zaposlenja",z.getDatum_zapolenja());
        podaci.put("telefon1",z.getTelefon1());
        podaci.put("telefon2",z.getTelefon2());
        db.update("Osoba",podaci,"OIB="+z.getOIB(),null);
        db.close();
    }
```

4.4.4.Ispis podataka

Dohvat i prikaz podataka zahtjeva nešto drugačiji pristup. Kako bi se podaci mogli pregledavat, potrebno ih je smisleno organizirati. U projektu su se za to koristile Liste koje su bile tipa nekog objekta, npr. *List<Osoba>, List<RadniNalog>,* itd.

Sami podaci su se dohvaćali putem standardnog SQL upita uz pomoć *.rawQuery* metode, a dohvaćeni podaci su se pohranjivali u varijablu tipa *Cursor*. Podaci u varijabli tipa *Cursor* su organizirani u redovima, onako kako su se dohvaćali iz baze podataka. Podaci se čitaju iz varijable red po red uz pomoć *while* petlje, gdje se odmah pohranjuju u listu, a na kraju se lista prosljeđuje kao rezultat izvršavanja metode. Podaci iz liste se onda mogu prikazivati na različite načine, stvaranjem zapisa u obliku znakovnog niza, unosom u padajuću listu ili pak u tablicu.

Primjer dohvata podataka Radnog naloga:

```
public List<RadniNalog> DohvatZaposlenikaPoRN()
{
      List<RadniNalog> listaRN=new ArrayList<RadniNalog>();
      SQLiteDatabase db = this.getWritableDatabase();
      Cursor c=db.rawQuery("SELECT rn.ID, z.ime, z.prezime,
      rn.datum_pocetka, rn.datum_zavrsetka, rn.vozilo,
      rn.dopunski_opis_rn \n" + "FROM Radni_nalog rn JOIN
      Zaduzene_osobe zo ON zo.radni_nalog=rn.ID\n" + "JOIN Osoba z ON
      zo.zaposlenik=z.OIB ORDER BY 1 asc, 3 asc",null);

         while (c.moveToNext())
         {
             Zaposlenik z=new Zaposlenik();
             z.setIme(c.getString(1));
             z.setPrezime(c.getString(2));
             RadniNalog rn=new RadniNalog();
             rn.setID(Integer.parseInt(c.getString(0)));
             rn.setDatum_od(c.getString(3));
             rn.setDatum_do(c.getString(4));
             rn.setVozilo(c.getString(5));
             rn.setOpis(c.getString(6));
             rn.setZap(z);
             listaRN.add(rn);
         }
         db.close();
         return  listaRN;
    }
```

U prethodno prikazanom isječku programskog koda bitno je napomenuti kako su podaci dohvaćeni, odnosno sortirani. Za organizaciju podataka je korištena naredba *Order by*, nakon naredbe slijedi broj koji označava stupac, u ovome slučaju je 1, što se odnosi na stupac *rn.ID*, a on je sortiran uzlazno (*ASC*). Ako se želi dalje specificirati kako se sortiraju podaci, onda se postavi zarez i navede se idući stupac i vrsta sortiranja. Na taj način se u jednom upitu mogu organizirati podaci na temelju više stupaca.

4.4.5.Brisanje podataka

Brisanje podataka je gotovo isto kao kod izmjene podataka. Može se koristiti *.execSQL* metoda gdje se upiše SQL upit ili se može koristiti *.delete* metoda, samo što onda nije potrebno organizirati podatke, već je samo potrebno specificirati gdje odnosno koji se podaci brišu.

Primjer brisanja Modela:

```
public void BrisiModel(Model mo)
    {
        SQLiteDatabase db=this.getWritableDatabase();
        db.execSQL("DELETE FROM Model WHERE ID="+mo.getID());
        db.close();
    }
```

To su svi načini upravljanja bazom podataka, a kroz čitavu implementaciju projekta su korištene manje varijacije gore navedenih isječaka koda, ovisno o samim podacima, tablicama te općenito potrebama.

4.5. Konačna izrada aplikacije

Dosad su opisani načini, te pojedinosti načina povezivanja i upravljanja bazom podataka, a u ovome dijelu će biti prikazana praktična primjena napisanog koda unutar aplikacije, odnosno na koji se način unose podaci, gdje i kako se čitaju ti isti podaci, itd. Dio upisanih podataka u bazi podataka će biti prikazan kroz nekoliko dijelova aplikacije, npr. padajućih listi i izbornika, dok će drugi dio biti prikazan putem tablica. Nadalje, veliki dio podataka je moguće i izmijeniti, no neke manje bitne podatke nije moguće ni brisati ni mijenjati zato jer se njima rijetko pristupa ili jednostavno nema potrebe za tim.

U sljedećem isječku koda će biti prikazan zaslon za obradu Radnog naloga, tj. unos. Gotovo sve što se u izradi aplikacije koristi je prikazano i implementirano ovdje. Na zaslonu su prikazana nekoliko polja, te dva padajuća izbornika. Polja i padajući izbornici predstavljaju atribute tablice „Radni nalog“ te se ti podaci ovdje i unose. Zatim se prelazi na idući zaslon gdje se za radni nalog dodaju usluge, a nakon toga slijedi unos zaposlenika koji su zaduženi za radni nalog.

U prvom dijelu koda unutar klase (aktivnosti) deklariraju se elementi koji se koriste na zaslonu ili unutar aplikacije, zatim slijedi inicijalizaciju pojedinih elemenata te adaptera koji se koriste unutar zaslona. Nakon toga slijedi stvaranje padajućih listi, odnosno njihovo punjenje, za koje postoje zasebne metode, (*PuniPadajuciIzbornikVozila()* i *PuniPadajućiIzbornikZaposlenicima()*). Obje metode su prikazane na kraju klase, te se samo pozovaju po potrebi.

```
public class UpravljanjeRadnimNalogom extends AppCompatActivity {

    private Spinner padajuciIzbornikVozila,padajuciVrsitelj;
    private EditText unosBrojVlasnika, unosOpisPosla,
    unosDatumaPocetka, unosDatumaZavrsetka;
```

```
    private Button gumbUnesi;
    private RadniNalogAdapter radniNalogAdapter;
    private VoziloAdapter voziloAdapter;
    private ZaposlenikAdapter zaposlenikAdapter;

    private DatePickerDialog picker;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_upravljanje_radnim_nalogom);

    padajuciIzbornikVozila=(Spinner)findViewById(R.id.padajuciIzbor
    nikVozila);

    padajuciVrsitelj=(Spinner)findViewById(R.id.padajuciVrsitelj);

    unosBrojVlasnika=(EditText)findViewById(R.id.unosBrojVlasnika);
    unosOpisPosla=(EditText)findViewById(R.id.unosOpisPosla);

    unosDatumaPocetka=(EditText)findViewById(R.id.unosDatumaPocetka
    );

    unosDatumaZavrsetka=(EditText)findViewById(R.id.unosDatumaZavrs
    etka);

    gumbUnesi=(Button)findViewById(R.id.gumbUnesi);

    radniNalogAdapter=new RadniNalogAdapter(this);
    zaposlenikAdapter=new ZaposlenikAdapter(this);
    voziloAdapter=new VoziloAdapter(this);
    unosDatumaZavrsetka.setShowSoftInputOnFocus(false);
    unosDatumaPocetka.setShowSoftInputOnFocus(false);

    PuniPadajuciIzbornikVozila();
    PuniPadajuciIzbornikZaposlenicima();

    ...
    }
}
```

Metode *PuniPadajuciIzbornikVozila()* i *PuniPadajućiIzbornikZaposlenicima()* se sastoje od sljedećeg programskog koda:

```
  private void PuniPadajuciIzbornikVozila()
  {
        ArrayAdapter<Vozilo> adapter=new
        ArrayAdapter<Vozilo>(this,R.layout.spinner_wraping_long_t
        ext,voziloAdapter.DohvatiSvaVozila());
```

```
        adapter.setDropDownViewResource(R.layout.spinner_wraping_
        long_text);
        padajuciIzbornikVozila.setAdapter(adapter);
    }

    private void PuniPadajuciIzbornikZaposlenicima()
    {
        ArrayAdapter<Zaposlenik> adapter=new
        ArrayAdapter<Zaposlenik>(this,R.layout.spinner_wraping_lo
        ng_text,zaposlenikAdapter.DohvatiSveZaposlenike());
        adapter.setDropDownViewResource(R.layout.spinner_wraping_
        long_text);
        padajuciVrsitelj.setAdapter(adapter);
    }
```

Zatim slijede pozivi događaja nad različitim elementima koji se nalaze na zaslonu pa se, primjerice, prilikom pritiska na polja za datum (*unosDatumaPocetka* i *unosDatumaZavrsetka*) otvara kalendar iz kojeg se može odabrati odgovarajući datum kako bi se izbjeglo pisanje datuma.

```
unosDatumaPocetka.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                final Calendar cldr = Calendar.getInstance();
                int day = cldr.get(Calendar.DAY_OF_MONTH);
                int month = cldr.get(Calendar.MONTH);
                int year = cldr.get(Calendar.YEAR);

                picker = new
                DatePickerDialog(UpravljanjeRadnimNalogom.this,
                        new DatePickerDialog.OnDateSetListener() {
                            @Override
                                public void onDateSet(DatePicker
                                view, int year, int monthOfYear,
                                int dayOfMonth) {
                                unosDatumaPocetka.setText(dayOfMont
                                h + "-" + (monthOfYear + 1) + "-" +
                                year);
                            }
                        }, year, month, day);
                picker.show();
            }
        });

        unosDatumaZavrsetka.setOnClickListener(new
        View.OnClickListener() {
            @Override
```

```
    public void onClick(View v) {
        final Calendar cldr = Calendar.getInstance();
        int day = cldr.get(Calendar.DAY_OF_MONTH);
        int month = cldr.get(Calendar.MONTH);
        int year = cldr.get(Calendar.YEAR);

        picker = new
        DatePickerDialog(UpravljanjeRadnimNalogom.this,
                new DatePickerDialog.OnDateSetListener() {
                    @Override
                         public void onDateSet(DatePicker
                         view, int year, int monthOfYear,
                         int dayOfMonth) {
                         unosDatumaZavrsetka.setText(dayOfMo
                         nth + "-" + (monthOfYear + 1) + "-"
                         + year);
                    }
                }, year, month, day);
        picker.show();
    }
});
```

Idući događaj je definiran nad gumbom *gumbUnesi*, gdje se pritiskom na taj isti gumb dohvaćaju upisani podaci, zatim se pripreme podaci za unos, pozove se metoda *UnosRadnogNaloga()* za unos podataka iz adaptera *radniNalogAdapter* te se podaci pohrane.

Nakon pohrane slijedi prijelaz na idući zaslon, gdje se proslijedi dodatna vrijednost (u ovom slučaju je to identifikator novounesenog radnog naloga). Za prijelaz na drugi zaslon (aktivnost) se koristi objekt *Intent* u koji se dohvati i pohrani željeni zaslon, a s *putExtra()* metodom se doda vrijednost koja se želi proslijediti na idući zaslon. Na kraju, nova aktivnost (zaslon) se pokrene uz pomoć metode *startActivity().*

```
gumbUnesi.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
   //Dohvaćanje podataka
        String broj=unosBrojVlasnika.getText().toString();
        String opis=unosOpisPosla.getText().toString();
        String
        datum_od=unosDatumaPocetka.getText().toString();
        String
        datum_do=unosDatumaZavrsetka.getText().toString();

   //Priprema podataka
```

```
                Vozilo
                vozilo=(Vozilo)padajuciIzbornikVozila.getSelectedIte
                m();
                Zaposlenik
                zaposlenik=(Zaposlenik)padajuciVrsitelj.getSelectedI
                tem();

                RadniNalog rn=new RadniNalog();
                rn.setOpis(opis);
                rn.setDatum_od(datum_od);
                rn.setDatum_do(datum_do);
                rn.setKontakt(broj);
                rn.setVoz(vozilo);
                rn.setZap(zaposlenik);

            //Unos unesenih podataka
                 long id=radniNalogAdapter.UnosRadnogNaloga(rn);
                 String idString=id+"";

            //Prijelaz na idući zaslon
                 Intent noviZaslon=new
                 Intent(getApplicationContext(),DodavanjeUslugaRadnom
                 Nalogu.class);

                 noviZaslon.putExtra("id_radni_nalog",idString);
                startActivity(noviZaslon);
              }
          });
```

Kako bi se dohvatili proslijeđeni podaci, koristi se metoda *hasExtra()*, gdje se upiše ključ te se provjeri jesu li proslijeđeni nekakvi podaci. U slučaju da jesu proslijeđeni podaci, onda se oni dohvaćaju pomoću metode *getExtras()*, a zatim se podaci pretvaraju u druge oblike po potrebi.

```
if(getIntent().hasExtra("id_radni_nalog")){
             String
             id=getIntent().getExtras().getString("id_radni_nalog");
             naslovRadniNalog.setText("RN - "+id);
             rnID=Integer.parseInt(id);
}
```

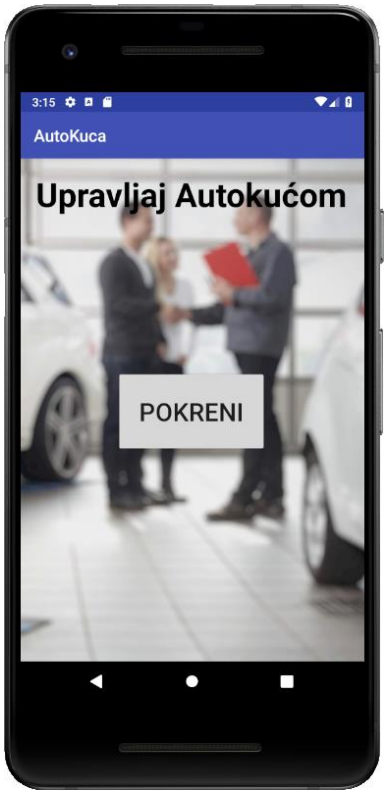
Na prikazan način je izrađena cijela aplikacija, jedina razlika u odnosu na druge zaslone (aktivnosti) leži u tome koji su to drugi događaji korišteni. Tako npr. imamo događaj *.addTextChangeListener* kod *EditText* elementa, *.setOnItemSelected Listener* kod *Spinner*-a itd. ovisno o potrebama, koristi se neki drugi događaj. U ovom su se dijelu mogli koristiti posebni vizualni elementi koji bi omogućili prilagođeniji prikaz i unos podataka.

5. Primjer korištenja

Konačnom aplikacijom je vrlo lako rukovati te ne zahtjeva nikakvog predznanja. Aplikaciju je potrebno instalirati i otvoriti, a sama aplikacija je osmišljena na način da korisnik iz izbornika odabere aktivnost koju želi izvršiti te se onda prijeđe na idući zaslon. Na samom zaslonu se nalaze polja i izbornici koja treba ispuniti i/ili iz kojih treba odabrati odgovarajuću opciju, a pritiskom na gumb se podaci unesu u bazu podataka.

U nekim slučajevima je moguće odabrati unesene podatke, također se pritisne na gumb („Izmjeni“) kako bi se mogli izmijeniti već neki uneseni podaci. Slično tome implementirano je i brisanje podataka, odabere se podatak i pritisne se gumb za brisanje.

Prilikom prvog pokretanja se otvara sljedeći zaslon:



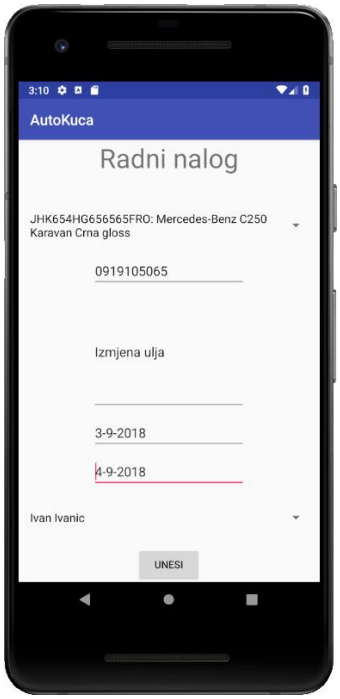
Slika 14. Početni zaslon (vlastita izrada)

Zaslon sa iznad prikazane slike (*Slika 14*) će se prilikom prvog pokretanja aplikacije sekundu duže otvarati jer je potrebno unijeti podatke o državama i gradovima u bazu podataka, a nakon toga svako iduće pokretanje se izvrši istog trenutka. Pritisne se na gumb “Pokreni“ čime se dolazi na glavni izbornik (Slika 16), a pritiskom na gumb „Upravljanje vozilima“ se otvara još jedan izbornik (Slika 15). Pritiskom gumba iz oba izbornika otvara se novi zaslon s formom za unos podataka i/ili ispis podataka.

Slika 16. Glavni izbornik (vlastita izrada)

Slika 15. Izbornik vozila (vlastita izrada)

U nastavku će biti prikazan unos Radnog naloga, usluga radnog naloga te zaposlenika koji su zaduženi za uneseni radni nalog.



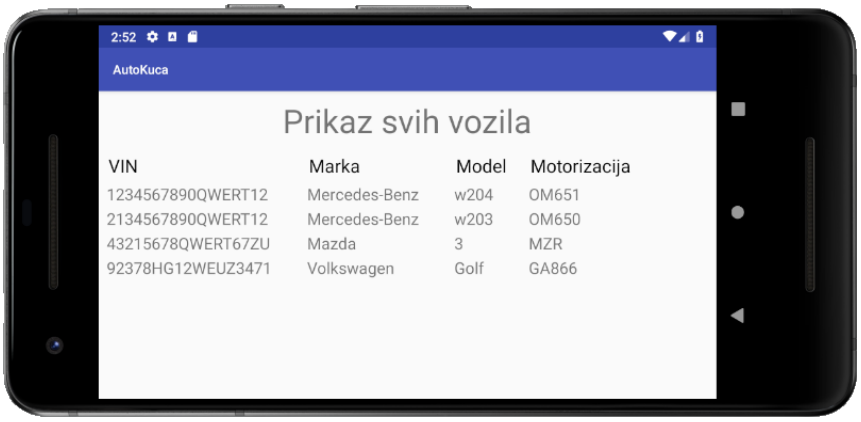
Slika 17. Unos radnog naloga (vlastita izrada)

Slika 18. Unos usluge RN (vlastita izrada)

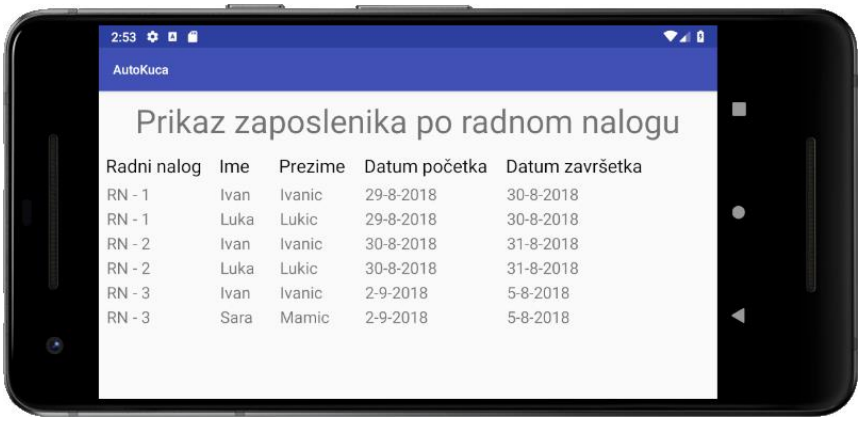
Slika 19. Unos zaduženih za RN (vlastita izrada)

Kod prvog zaslona (*Slika 17*) se unose osnovni podaci za radni nalog, tj. oni atributi koji su navedeni u tablici „Radni_nalog“. Podaci koji su već uneseni ili koji postoje se biraju iz padajućih izbornika, kako bi krajnjem korisniku korištenje aplikacije bilo što jednostavnije. Pritiskom na gumb „Unesi“ se kreira novi radni nalog te se identifikator tog radnog naloga prenosi kao parametar na idući zaslon (*Slika 18*). Na drugom zaslonu se unose sve usluge koje će se izvršiti za proslijeđeni radni nalog te se računa ukupna cijena, a zatim se pritisne gumb „Dalje“ kako bi se prešlo na posljednji zaslon (*Slika 19*). Na zadnjem zaslonu se unosi zaposlenik ili zaposlenici koji su zaduženi za radni nalog, te je to zadnji korak kod kreiranja radnog naloga.

Također postoji prikaz podataka u obliku tablica, jedna tablica prikazuje sva unesena vozila, a druga tablica pokazuje koji radni nalog koji zaposlenik obavlja i od kad do kad se izvršava radni nalog.



Slika 18. Prikaz vozila (vlastita izrada)



Slika 19. Prikaz zaposlenika po radnom nalogu (vlastita izrada)

6. Zaključak

Izrada SQLite baze podataka je samo po sebi prilično jednostavna, uključujući okidače. Potrebno je znati nekoliko naredbi uz pomoć kojih se kreiraju tablice, a time i baza podataka, drugi način bi bilo korištenje nekakvog grafičkog sučelja gdje je kreiranje tablica, okidača i pogleda znatno jednostavnije i preglednije, a i samo upravljanje tako kreirano bazom je dosta jednostavnije. Što se SQLite-a tiče, ona je vrlo korisna i jednostavna relacijska baza podataka, posebice zbog malog broja tipova podataka i jednostavnosti pisanja okidača. Unatoč tome što sadrži mali broj funkcionalnosti, SQLite je i dalje vrlo moćan, pogotovo kod web servisa i manjih mobilnih aplikacija, sve što mu se suprotstavi može izvršiti brzo i efikasno.

Što se tiče izrade aplikacije, ono je znatno kompliciranije i zahtjevnije. Aplikacija koja je u radu prikazan je vrlo jednostavna, no iza nje stoji dosta posla, a tek se dotaklo ono najosnovnije. Java okruženje i Android studio su vrlo korisni alati i s njima se može znatno više napraviti. Napomenuo bih da je kreiranje SQLite baze podataka unutar Jave dosta nespretno napravljeno, potrebno je pisati duge upite, gdje se lako može pogriješiti, u tom slučaju bi bilo znatno jednostavnije da se učita gotova baza podataka (.db datoteka) te se iz nje kreiraju tablice čime bi se ubrzalo kreiranje baze podataka u Javi.

Jedino što bih dodao u aplikaciju su posebno kreirani elementi za pregled i izmjenu podataka, poput, primjerice, pokretnih lista, gdje bi se na svakoj pločici nalazilo npr. jedno vozilo, a klikom na vozilo prikazala bi se mogućnost izmjene vozila, brisanje vozila i premještaj vozila.

Aplikacija je korektno odrađena, te se vrlo jednostavno i intuitivno koristi, ali se takav tip teme mogao drugačije realizirati. Primjer auto kuće bi se razradio na uređaju s većim zaslonom ili klasičnom računalu gdje bi glavni sustav za upravljanje bazom podataka bio MySQL koji bi se koristio u kombinaciji s SQLite radi boljih performansi i lakšeg upravljana nad podacima. Sami sustav SQLite bi bio korisniji kod manje aplikacije poput *Note* aplikacije za pisanje bilješki ili slično.

7. Literatura

[1] Kornelije R., (2011.) Uvod u SQL, Fakultet organizacije i informatike, Varaždin

[2] Kornelije R., (2014.) SQL – Napredne teme, Fakultet organizacije i informatike, Varaždin

[3] Kreibich, J. R (2010.) Using SQLite, O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472

[4] Allen, G. i Owens, M. (2010.) The Definitive Guide to SQLite, Secon Edition, Apress, USA, New York: Springer Science+Business Media

[5] SQLite, (2018.) About SQLite, Preuzeto 03.09.2018. s https://www.sqlite.org/about.html

[6] Navicat, (2018.) Navicat, Preuzeto 05.09.2018. s https://www.navicat.com/en/products/navicat-premium

[7] Developer, (2018.) Meet Android Studio, Preuzeto 03.09.2018 s https://developer.android.com/studio/intro/

[8] Oracle, (2018.) What is JAVA, Preuzeto 03.09.2018. s https://www.java.com/en/download/faq/whatis_java.xml

[9] DigitalOcean, (2018.) SQLite vs MySQL vs PostgreSQL: A Comparison of Rational Database Management Systems, Preuzeto 03.09.2018 s https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

[10] H3RALD, (2018.) A Qick Overview of SQLite, Preuzeto 04.09.2018. s https://h3rald.com/articles/quick-overview-of-sqlite/

[11] Radošević, D. , (2007.), Programiranje 2, TIVA Tiskara: Fakultet organizacije i informatike, Varaždin

[12] Developer (2018). Android Studio (Verzija 3.1.4), Preuzeto s https://developer.android.com/studio/

[13] Navicat (2018). Navicat for SQLite (Verzija 12.1), Preuzeto s https://www.navicat.com/en/download/navicat-for-sqlite

[14] DB Browser (2018). DB Browser for SQLite (Verzija 3.10.1), Preuzeto s https://sqlitebrowser.org

8. Popis korištenih slika

9. Prilog

1. Izvorni kod, projekt i aplikacija: https://mega.nz/#F!qT4TkZQK!frNFV-5TQGxDz06mSBNR3w